5

- *Qt Aletlerinin Hususileştirilmesi*
- *QWidget dan altsınıf oluşturulması*
- *Hususi Aletlerin Qt Designer Proğramına Eklenmesi*
- *Çifte Arabellek (Double Buffering)*

Özel Aletlerin Oluşturulması

Bu bölümde hususi Qt aletlerinin oluşturulmasını göreceğiz. Hususi aletler mevcut Qt aletlerinden birisinden alt sınıf oluşturmak suretiyla veyahutta direk olarak QWidget dan alt sınıf oluşturmak suretiyle yapılabilirler. Her iki yaklaşımıda göreceğiz. Ayrıca bu hususi aletlerin Qt Designer altında mevcut aletler giba kullanılabilmeleri için gerekli adımlarıda göstereceüiz. Bu bölümü ekrandaki titreşimleri (flicker) önleyen hususi bir aletin tasarımı ile sona erdireceğiz ki bu alet fevkalade bir teknik olan çifte arabellek (double buffering) istimal etmektedir.

Qt Aletlerinin Özelleştirlmesi

Bazı durumlarda bir Qt aletinin özelleştirilmesi için Qt Designer içerisindeki mevcut seçenekler veya bu aletin üye fonksiyonlarının sunduğu seçenekler yetersiz kalmaktadır. Bu problemin basit ve direk bi çözümü sözkonusu aletin alt sınıfını oluğturup gerekli değişiklik veya ilaveleri alt sınıf iöerisinde yapmaktır.



Şekil 5.1: HexSpinBox aleti

Bu kısımda, özelleştirmenin nasıl yapılabileceğini göstermek için, bir hexadecimal fırıldak kutusu (spin box) geliştireceğiz. QSpinBox sadece decimal tamsayılar (integers) gösterir, fakat bundan alt sınıf oluşturarak hexadecimal değerleri güstermesini sağlamak gayet kolaydır.

```
#ifndef HEXSPINBOX_H
#define HEXSPINBOX_H
#include <qspinbox.h>
class HexSpinBox : public QSpinBox
{
public:
      HexSpinBox(QWidget *parent,const char *name=0);
protected:
```

```
        QString mapValueToText(int value);
        int mapTextToValue(bool *ok);
};

#endif
```

HexSpinBox sınıfının yeteneklerinin çoğu ona QSpinBox sınıfından miras kalmıştır. Bu sınıfın tipik bir yapıcısı olup QSpinBox sınıfının iki hayali fonksiyonunu yeniden tanımlar. Bu sınıfın kendisine has sinyal ve dilimleri olmadığından Q_ OBJECT makrosuna gerek yoktur.

```
#include <qvalidator.h>
#include "hexspinbox.h"
HexSpinBox::HexSpinBox(QWidget *parent,const char*name)
        :QSpinBox(parent, name)
{
        QRegExp regExp("[0-9A-Fa-f]+");
        setValidator(new QRegExpValidator(regExp, this));
        setRange(0, 255);
}
```

Kullanıcı fırıldak kutusunun değerini aiağı yukarı oklarına tıklamak veya klaviyeden (onun satır muharririne) yazmak suretiyle değiştirebilir. Kullanıcının kılaviyeden yazması durumunda sadece geçerli hexadecimal değerleri girmesine izin vereceğiz. Bunu başarmak içim bir QRegExpValidator kullanacağoz ki bu müteberperver (validator) sadece '0'-'9' , 'A' -'F' , ve 'a'-'f' arasındaki değerleri kabul eder. Normalde bu aletin sınırlarını 0 ila 255 (0x00 ila 0xFF) olarak ayarlıyoruz ki bu sınırlar hexadecimal bir fırıldak kutusu için QSpinBox ın normal sınır değerleri olan 0 ve 99 den daha makuldürler.

```
QString HexSpinBox::mapValueToText(int value)
{
        return QString::number(value, 16).upper();
}
```

mapTextToValue() fonksiyonu metini tamsayıya çevirir. Kullanıcı QSpinBox ın satır editörüne yazıp Enter tuğuna bastığında bu fonksiyon çağrılır. Qstring::toInt() fonksiyonunu kullanarak QspinBox::text() tarafından geri gönderilen metni 16 tabanına gğre tam sayıya çevirmeye çalışıyoruz, eğer çeviri başarı ile tamamlandı ise toInt() fonksiyonu ok değişkeninin değerini müsbet, aksi taktirde menfi olarak tesbit eder. Bu davranış QSpinBox beklentisinin ta kendisidir.

Böylece bir fırıldak kutusunun özelleştirilmesini tamamlamış olduk. Sair Qt aletlerinin hususileştirilmeleride aynı adımları takip eder: Müsait bir Qt aleti seç, ondan bir alt sınıf oluştur, ve davranışını değiştirmek için bazı hayali fonksiyonları yeniden tanımla. Bu metod Qt proğramlamada gayet yaygındır; aslında biz bu tekniği dördüncü bölümde QTable sınıfından bir alt sınıf oluşturup createEditor() ve endEdit() fonksiyonlarını yeniden tanımlamakla kullanmış idik.

QWidget dan Altsınıf Oluşturulması

Özel aletlerin çoğu, ister Qt de mevcut olanlar olsun isterse HexSpinBox gibi kullanıcı tarafından yapılmış olsun, aslında hali hazırdaki aletlerin mexcedilmesi yada bir araya getirilmei ile oluşturulmuşlardır. Mevcut aletlerin bırleştirilmesi ile yapılan aletler genellikle Qt Designer içerisinde oluşturulabilirler:

- Widget kalıbını (template) kullanarak yeni bir form oluştur.
- Gerekli aletleri forma ekle ve (dizim mekanizmasını kullanarak) onları form üzerine yerleştir.
- Gerekli sinyal ve dilim bağlantılarını kur ve özel aletin uygun şekilde işlemesi için lüzümlu olan kodu ya bir .ui.h dosyası içerisinde yada bir alt sınıfta tedarik et.

Tabiiki bunun tamamı Qt Designer dişinda kod yazmak suretiylede gerçekleştirilebilir. Hangi yaklaşım kullanılırsa kullanılsın sonuçta elde edilen sınıf QWidget sınıfının direk varisi olur.

Eğer yeni aletin kendine has sinyal ve dilimleri yoksa ve hiç bir hayali fonksiyonu yeniden tanımlamıyorsa bu aleti aletlerin basitce biraraya getirilmesi ile alt sınıf oluşturmadan yapmak mümkündür. Bu yaklaşımı, birinci bölümde Yaş proğramını, bir QHBox, bir QSpinBox ve birde QSlider aleti kullnarak oluştururken istimal etmişdik. İsteseidik QHBox san alt sınıf oluşturabilirdik ve bu alt sınıfın yapıcısı içerisinde QSpinBox ve QSlider nesnelerini oluşturabilirdik.

Mevcut Qt aletlerinden hiç birisi arzu edilen yeteneklere sahip değil ise ve yine hali hazırdaki aletlerin cem edilmeleri istenilen neticeye vermeyecek ise bu durumda dahi istediğimiz aleti oluşturabiliriz. Bunu QWidget dan alt sınıf oluşturup aleti görüntülemek ve fare tıklamalarına cevap vermesi için bir kaç eylem halledicileri (event handlers) yeniden tanımlamak suretiyle gerçekleştirebiliriz. Bu yaklaşım aletin görüntülenmesinden davranışlarına kadar bütün yönleriyle kontrolünü proğramcının eline verir. Qt nin mevcu altlerinden QLabel, QPushButton, QTable gibileri bu şekilde oluşturulmuşlardır. Şayet bu aletler Qt de mevcut olmasalar idi onları QWidget tarafından tedarik edilen umumi (public) fonksiyonlar vasıtasıyla bu aletleri işletim sisteminden bağımsız bir şekilde oluşturmak mümkün olurdu.

Özel aletlerin nasıl yazılabileceğini göstermek için şekil 5.2 de gösterilen IconEditor aletini oluşturacağız. IconEditor aleti simge düzenleme proğramında kullanılabilecek bir alettir.

Başlık dosyasını gözden geçirerek işe başlayalım.

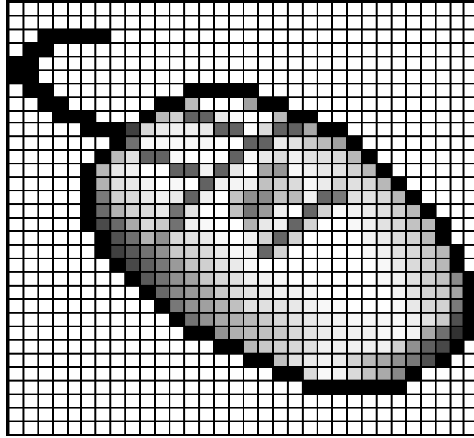
```
#ifndef ICONEDITOR_H
#define ICONEDITOR_H
#include <qimage.h>
#include <qwidget.h>
class IconEditor : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(QColor penColor READ penColor WRITE setPenColor)
```

```
        Q_PROPERTY(QImage iconImage READ iconImage WRITE setIconImage)
        Q_PROPERTY(int zoomFactor READ zoomFactor WRITE setZoomFactor)

    public:
        IconEditor(QWidget *parent = 0, const char *name = 0);

        void setPenColor(const QColor &newColor);
        QColor penColor() const { return curColor; }
        void setZoomFactor(int newZoom);
        int zoomFactor() const { return zoom; }
        void setIconImage(const QImage &newImage);
        const QImage &iconImage() const { return image; }
        QSize sizeHint() const;
```

IconEditor sınıfı Q_PROPERTY() makrosunu kullnarak üç tene hususi özellik tanımlamaktadır: penColor[1], iconImage[2], and zoomFactor[3]. Her özelliğin bir türü, bir okuma fonksiyonu ve birde yazma fonksiyonu mevcuttur. Mesela, penColor özelliği QColor türünden olup penColor()[4] ve setPenColor()[5] fonksiyonları kullanılarak bu özelliği okuyup yazılabilir[6].



Şekil 5.2: IconEditor aleti

Qt Desiğner içerisinde bir aletin hususi özellikleri QWidget sınıfından miras kalan özelliklerden sonra listelenirler ve bu özelliklerin türü QVariant tarafından desteklenen herhangi bir türde olabilir. Q_OBJECT makrosunu kullanmak özellik tanımlayan sınıflar için elzemdir.

1 kalem rengi
2 simge resmi/imajı
3 büyütme/küçültme faktörü
4 Kalemib rengini getir
5 Kalemin rengini ayarla/tesbit et
6 Burada okumaktan maksat bu değere ulaşmak ve yazmaktan maksat ise bu değeri değiştirmektir.

```
protected:
   void mousePressEvent(QMouseEvent *event);
   void mouseMoveEvent(QMouseEvent *event);
   void paintEvent(QPaintEvent *event);

private:
   void drawImagePixel(QPainter *painter, int i, int j);
   void setImagePixel(const QPoint &pos, bool opaque);
   QColor curColor; QImage image; int zoom;
};
#endif
```

IconEditor, QWidget sınıfının üç mahfuz (protected) sınıfını yeniden tanımlar ve kendisine has bir takım hususi (private) fonksiyonları ve değişkenleri mevcuttur. Bu üç hususi deüişken sınıfın üç özelliğini tutmaktadır. Sınıfın kodunu ihtiva eden dosya #include komutu ile başlar ve yapıcı ile devam eder:

```
#include <qpainter.h>
#include "iconeditor.h"
IconEditor::IconEditor(QWidget *parent, const char *name)
      :QWidget(parent, name, WStaticContents)
{
   setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Minimum);
   curColor = black;
   zoom = 8;
   image.create(16, 16, 32);
   image.fill(qRgba(0, 0, 0, 0));
   image.setAlphaBuffer(true);
}
```

Yapıcının, setSizePolicy() fonksiyonunu çağırması ve WStaticContents seçeneği kayga değer hususlardır. Birazdan bunları ele alacağız.

Büyütme faktörü (zoom) 8 olarak ayarlanmıştır, bu demek oluyorki simgedeki her bir piksel 8x8 büyüklüğündeki bir kare olarak gösterilecektir. Kalemin rengi siyah (black) olarak ayarlanmıştır; “black” sembolü QObject sınıfının üst sınıfı olan Qt sınıfında tanımlanmıştır.

Simgeye ait data “image” değişkeninde tutulmaktadır ve bu dataya setIconImage() ve iconImage() fonksiyonlarını kullanarak ulaşılabilir. Bir simge editör proğramı normalde kullanıcı bir simge yüklediğinde setIconImage() fonksiyonunu ve simgeyi kaydetmek istediğinde ise iconImage() fonksiyonunu kullanır.

“image” değişkeni Qimage türündendir. Bu değişkeni başlangıçta 16 × 16 piksele ve 32-bit derinliğine ayarlıyor, resim değişkenini temizliyor ve “alpha buffer” acrif hele getiriyoruz.

QImage sınıfı resmi donanımdan bağımsiz bir şekilde tutar. İstenildiğinde 1-bit, 8-bit, veya 32-bit derinliğe ayarlanabilir. 32-bit derinliğindeki bir resim bir pikselin kırmızı, yeşil ve mavi bileşenlerinden her biri için 8 bit kullanır. Geri kalan 8 bit pikselin alfa (alpha) bileşenini, yani şeffaflık derecesini, ihtiva eder. Mesela, saf kırmızı renk için bu dört bileşenin, yani kırmızı, yeşil, mavi ve alfa, değerleri sıtasıyla şöyledir: 255, 0, 0, ve 255. Qt altınada bu renk ya şu şekilde

```
QRgb red = qRgba(255, 0, 0, 255);
```

veya

```
QRgb red = qRgb(255, 0, 0);
```

şeklinde oluşturulabilir. QRgb aslında `unsigned int` için bir `typedef` dir, ayrıca qRgb() ve qRgba() satır arası (inline functions) olup argümanlarını 32-bit integer değerinde toplarlar. Şu şekilde yazmakta mümkündür:

```
QRgb red = 0xFFFF0000 ;
```

ki burada ilk FF alfa bileşenine, ikinci FF ise kırmızı bileşenine tekabül eder. IconEditor ün yapıcısında Qimage, alfa değeri olarak sıfırı (0) kullanmak suretiyle, tam şeffaf bir renkle doldurulur. Qt de renk ya QRgb veya Qcolor olarak muhafaza edilir. Bir typedef olan QRgb, 32-bit resimlerin tutulması için sadece QImage tarafından kullanılırken, QColor sınıfı bir çok kullanışlı üye fonksiyonları tedarik eder ve Qt içerisinde sıkça kullanılır. IconEditor aletinde QRgb yi sadece QImage ile çalışırken kullanıyoruz; geri kalan her şey için QColor kullanıyoruz ki buna penColor özelliği de dahildir.

```
QSize IconEditor::sizeHint() const
{
      QSize size = zoom * image.size();
      if (zoom >= 3) size += QSize(1, 1);
      return size;
}
```

QWidget sınıfının sizeHint() fonksiyonu yeniden tanımlandı ve o aletin ideal büyüklüğünü yada ebatlarını üretir. Burada resmin büyüklüğünü alıyor büyütmr faktörü ile çarptıktan sonra,eğer büyütme faktörü 3 den büyük ise, her yöde bir ziyade pikdel ekliyoruz (izgara için) (Büyütme faktörü 2 veya bir ise izgarayı görüntülemiyoruz çünkü girid simgenin bütün piksellerini kaplar.) Bir aletin tavsiye edilen büyüklüğü (size hint) en çok dizim mekanizması için ehemmiyet arzeder. Qt nin dizim mekanizması formun çocuklarını form üzerine yerleştirirken mümkün mertebe tavsiye edilen ebatları kullanmaya çalışırlar. IconEditor ün diziminin başarılı olabilmesi için güvenilir yada makul ebat tavsiye etmesi gerekir.

Aletler, ideal ebatlarına ilaveten dizim mekanizmalarına ebat tarzları (size policy) hakkındada bilgi verirler ki bu onların gerilme yada büzülme isteyip istemediklerini dizim mekanizmalarına bildirir. Yapıcı içerisinde,düşey ve yatay ebat tarzları için, QSizePolicy::Minimum ile setSizePolicy() fonksiyonunu çağırmak suretiyle dizim mekanizmalarina aletin tavsiye edilen büyüklüğünün maku olacak en küçük büyüklük olduğunu bildirmiş oluyoruz. Bir başka deyişle bu latin gerektiğinde gerilebileceğini ancak tavsiye edilen abatın altına inecek şekilde asla büzülmemelidir. Aletin bu davranışı Qt Designer içerisinde sizePolicy özelliğini değiştirmek suretiyle tebdil edilebilir. Müteaddid ebat tarzları altıncı bölümde (Dizim Mekanizmalar) tafsilen beyan edileceklerdir.

```
void IconEditor::setPenColor(const QColor &newColor)
{
      curColor = newColor;
}
```

The setPenColor() function sets the current pen color. The color will be used for newly drawn

pixels.

```
void IconEditor::setIconImage(const QImage &newImage)
{
      if (newImage != image)
      {
            image = newImage.convertDepth(32);
            image.detach();
            update();
            updateGeometry();
      }
}
```

setIconImage() fonksiyonu değiştirilecek olan resmi belirler. Eşer reism 32-bit değil ise convertDepth()[7] fonkisonunu çağırarak onu 32-bit derinliğine çeviriyoruz. Proğramın diğer kısımlarında resmin 32-bit QRgb olarak saklandığını varsayacağız. image deüişkeninde tutulan datanın tamamen kopyalanması için detach() fonksiyonunu çağırıyoruz. Bunu yapmamızın sebebi resim datasının ROM[8] da tutlma ihtimalidir. QImage zaman ve bellekten tasarruf etmek için datayı ondan istenildiği an kopyalar. Bu tür optimizasyona "explicit sharing[9]" adı verilir ve QMemArray<T> ile müşire dayalı muhtevi (Pointer-Based Containers) kısmında 11. bölümde ele alınacaktır.

image değişkenine arzu edilen resmi yerleştirdikten sonra QWidget::update()[10] fonksiyonunu çağırmak suretiyle yeni resmin ekranda gösterilmesini (daha doğrusu remin IconImage aleti üzerinde gösterilmesini) sağlarız. Daha sonra, QWidget::updateGeometry()[11] fonksiyonunu çağırmak suretiyle bu aleti ihtiva eden dizim veya dizimlere aletin tavsiye edilen ebatlarının değiştiğini bildiririz. Dizim artık bu yeni ideal ebata göre aletin boyutlarını ayarlar.

```
void IconEditor::setZoomFactor(int newZoom)
{
      if (newZoom < 1)
            newZoom = 1;
      if (newZoom != zoom)
      {
            zoom = newZoom;
            update();
            updateGeometry();
      }
}
```

setZoomFactor() fonksiyonu resmin büyütme faktörünü ayarlar. İleride sıfıra bölme hadisesinin vuku bulmaması için birden küçük olan büyütme faktörünü bir olarak düzeştiyoruz. Yine update() ve updateGeometry() fonksiyonlarını çağırarak aleti güncelleştiriyor ve aleti ihtiva eden dizimi ideal ebat değişikliği hususunda bilgilendiriyoruz.

---

7 derinliğini değiştir

8 Read Only Memory (yanlız okunabilen bellek)

9 vazih paylaşma

10 güncelleştir

11 geometriyi güncelleştir

penColor(), iconImage() ve zoomFactor() fonksiyonları başlık dosyasında satırarası fonksiyon (inline functions) olarak tanımlandı.

Şimdi paintEvent() fonksiyonunun kodunu gözden geçireceğiz. Bu fonksiyon IconEditor aletinin en mühim fonksiyonudur. Ne zamanki aletin yeniden çizilmesi (boyanması veye güncelleştirilmesi) gerekiyorsa bu fonksiyon çağrılır. Aslında QWidget sınıfından miras kalan bu fonksiyon orjinal hali ile hiö bir şey yapmaz ve aleti boş bırakır.

Üçüncü bölümde karıılaştıüımız contextMenuEvent() ve closeEvent() fonksiyonları gibi , paintEvent() fonksiyonuda bir eylem helledecidir (event handler). At de her biri değişik bir eyleme cevap veren çok sayıda eylem halledicileri mevcuttur. Yedinci bölümde eylemleri detaylı olarak göreceğiz.

Boya eyleminin (paint event) hsule gelmesini ve paintEvent() fonksiyonun öağrılmasını gerektiren haller:

- Alet ilk kez gösteriliyor ise, işletim sistemi otomatik olarak boya eylemi husule getirir ve aletin kendini boyamasını elzem kılar.
- Aletin ebatları değiştirildiğinde, işletim sistemi otomatik olarak boya eylemi husule getirir.
- Eğer bir alet kısmen veya tamamen bir başka alet tarafından örtülmüş ise, pencere sisyemi örtülmiş alanı muhafaza etmemiş ise, bu kısım için boya eylemi husule getirilir.

Biz istersek QWidget::update() veya Qwidget::repaint() fonksiyonunu çağırmak suretiyle boya eylemi husuşe getirebiliriz. Bu iki fonksiyon arasındaki fark, repaint() fonksiyonunun derhal aletin boyanmasını zorunlu kılarken update() fonksiyonunun boyama işlemi için “randevü” alması ve boyama işleminin bir sonraki Qt eylem halletme işlemi sırasında gerçekleştirilmesidir. (Alet ekranda görünür vaziyette değil ise her iki fonksiyon hiç bir şey yapmazlar) Eğer update() fonksiyonu çok defa ve sıklıkla çağrılırsa, Qt bu eylemleri birleştirip yanlızca bir boyama yapar taki ekranda oluşabilecek mutemel titreşimi önlesin. IconEditor için biz daima update() fonksiyonunu kullanıyoruz.

İşte kod:

```
void IconEditor::paintEvent(QPaintEvent *)
{
      QPainter painter(this); if (zoom >= 3)
      {
            painter.setPen(colorGroup().foreground());
            for (int i = 0; i <= image.width(); ++i)
                  painter.drawLine(zoom * i, 0,
                              zoom * i, zoom * image.height());
            for (int j = 0; j <= image.height(); ++j)
                  painter.drawLine(0, zoom * j,
                                 zoom * image.width(), zoom * j);
      }
      for (int i = 0; i < image.width(); ++i)
      {
            for (int j = 0; j < image.height(); ++j)
```

```
                        drawImagePixel(&painter, i, j);
            }
      }
```

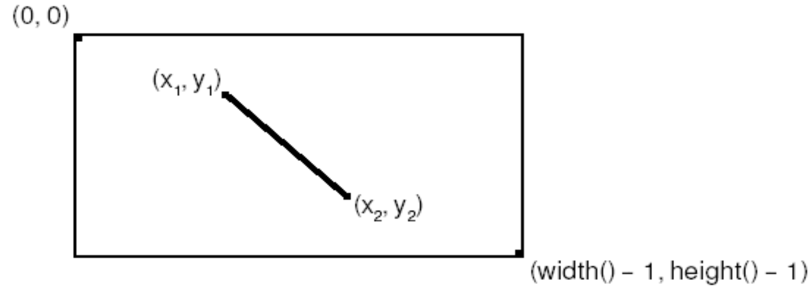
Alete mahsus bir QPainter nesnesi oluşturarak işe başlıyoruz. Eğer büyütme faktörü 3 va ya daha büyük ise, izgarayı oluşturacak olan yatay ve düşey çizgileri QPainter::drawLine() fonksiyonunu kullnarak öiziyoruz.

QPainter::drawLine() fonksiyonu şu şekilde çağrılır:

```
painter.drawLine(x1, y1, x2, y2);
```

Burada (x1, y1) noktası çizginin bir ucunuun pozisyonunu ve (x2, y2) noktası ise diğer ucunun pozisyonunu verir. Bu fonksiyonun bir başka versiyonuda mevcuttur ki bu dört tane integer yerine iki tane QPoint türü değişken alır.

Qt aletlerinin sol üst köşesindeki pikseli (0, 0) pozisyonunda ve ,sağ alt köşesindeki pikseli ise (width()[12] -1, height()[13] -1) pozisyonunda yer almaktadır. Bu konvansiyonel kartezyen kordinate sistemine çok benzer ancak y ekseni aşaği doğru yönlendirilmişdir ki buda GUI proğramlamaya gayet müsaittir. QPainter sınıfının kordinat sistemini transformasyon (kaydırma, büyütme/küöültme, dönderme ve kesme gibi) kullanarak değiştirmek mümkündür. Bu hsuus sekizinci bölümde (İki Boyutlu ve Üç Boyutlu Grafikler) ele alınacaktır.



Şekil 5.3: QPainter kullanarak çizgi çizimi.

QPainter in drawLine() fonksiyonunu çağırmadan önce setPen() fonksiyonunu kullanarak kalemin rengini ayarlıyoruz. Siyah veya gri gibi her hangi bir sabit renk kullanabiliriz ancak daha iyiy yaklaşım aletin boya paleti sini kullanmak olur.

Her bir aletin hangi rengin hangi iş için kullanılacağını belirleyen bir boya paleti mevcuttur. Mesela, boya paletinde aletlerin arakaplan renklerini belirlemek için bir girdi (genelde açık gri) ve yazı rengi (genelde siyah)için bir girdi mevcuttur. Normalde aletin boya paleti pencere sisteminin renk paketini kullanır. Boya paltindeki renkleri kullanmakla IconEditor kullanıcının renk tercihine saygılı olmasını sağlamış oluruz. Bir aletin boya paleti aktif, pasif ve malul (disabled) diye üç renk gurubu mevcuttur. Bunlardan hangisinin kullanılacağı aletin o anki haline bağlıdır:

- Aktif renk gurubu aletin aktif pencerede yer alması halinde kullanılır.

12 bu fonksiyon aletin genişliğini verir

13 bu fonksiyon aletin yüksekliğini verir

- Pasif renk gurubu aletin diğer pencerelerde yer alması halinde kullanılır.
- Malul renk gurubu ise malul edilmiş olan aletler için kullanılı.

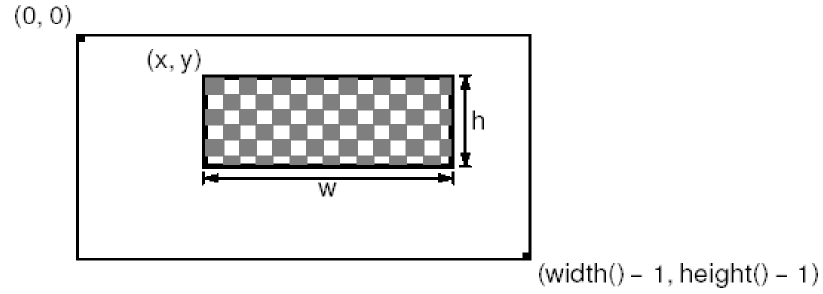
QWidget::palette() fonksiyonu aletin paletini QPalette türünde bir nesne olarak verir. QPalette nin değişik renk guruplarına active(), inactive() ve disabled() fonksiyonlarını kullanarak ulaşmak mümkündür ki bu renk gurupları QcolorGroup türündendirler. Kolaylık olması bakımından QWidget::colorGroup() fonksiyonu aletin hali hazırdaki durumunu göz önüne almak suretiyle bu hale müsait renk gurubunu verir, bu demek oluyorki bizim renk paletine direk olarak ulaşmamıza nediren ihtiyaç duyulur.

paintEvent() fonksiyonu resmin çizimini kendisi tamalar ve bu işlem için IconEditor::drawImagePixel() fonksiyonundan faydalanır ki bu fonksiyon resmin her bir pikselini dolmiş kare olarak görüntüler.

```
void IconEditor::drawImagePixel(QPainter *painter, int i, int j)
{
      QColor color;
      QRgb rgb = image.pixel(i, j);
      if (qAlpha(rgb) == 0)
            color = colorGroup().base();
      else
            color.setRgb(rgb);
      if (zoom >= 3)
      {
            painter->fillRect(zoom * i + 1, zoom * j + 1,
                              zoom - 1, zoom - 1, color);
      }
      else
      {
            painter->fillRect(zoom * i, zoom * j,
                              zoom, zoom, color);
      }
}
```

drawImagePixel() fonksiyonu büyültülmüi pikseli QPainter kullanarak çizer. i ve j parametreleri pikselin QImage içerisindeki kordinatları olup alet üzerindeki kordinatları değillerdir. (Büyütme faktörü 1 ise her iki kordinat bir birinin aynısıdır) Eğer piksel tamamaen şeffaf ise (yani alfa değeri 0), o anki renk gurubunun temel rengini (bu genelde beyazdır) kullanarak pikseli çiziyoruz, dişer durumlarda pikselin resimdeki rengini kullanıyoruz. Sonra QPainter::fillRect() fonksiyonunu kullanarak dolu kare çiziyoruz. Eğer izgara aktif ise yada görüntüleniyor ise pikselin boyutunu her iki yönde bir küçültüyoruzki izgarayı kapatmayalım.

Şekil 5.4: QPainter kullanarak dikdörtgen çizimi.



QPainter::fillRect() fonksiyonu şu şekilde çağrılabilir:

```
painter->fillRect(x, y, w, h, brush);
```

Burada (x, y) noktası dikdörtgenin sol köşesinin pozisyonun, w genişliğini, h yüksekliğini ve brush (fırça) kullanılacak rengi ve deseni belirler. QColor türünde bir nesneyi "brush" olarak kullanmak suretiyle dikdörtgeni desensiz bir şekilde dolduruyoruz.

```
void IconEditor::mousePressEvent(QMouseEvent *event)
{
      if (event->button() == LeftButton)
            setImagePixel(event->pos(), true);
      else if (event->button() == RightButton)
            setImagePixel(event->pos(), false);
}
```

Kullanıcı fare tuşuna bastığında işletim sistemi fareye basıldı eylemi (mouse press event) husule getirir. Qwidget::mousePressEvent() fonksiyonunu yeniden tanımlamak suretiyle, fare tuğuna basılması hadiselerine fare imlecinin altındaki pikseli temizlemek veya boyamak suretiyle karşılık verebiliriz.

Kullanıcı sol fare tuğuna bastıysa, ikinci argümanı müsbet (true) olacak şekilde setImagePixel() hususi fonksiyonunu çağırıyoruz ve ona imlecin altındaki pikseli o anki kalem rengine çevirmesi talimatını veriyoruz. Kullanıcı sağ fare tuğuna bastı ise biz yine setImagePixel() fonksiyonunu çağırıyoruz ama bu defa pikseli temizleme talimatını veriyoruz.

```
void IconEditor::mouseMoveEvent(QMouseEvent *event)
{
      if (event->state() & LeftButton)
            setImagePixel(event->pos(), true);
      else if (event->state() & RightButton)
            setImagePixel(event->pos(), false);
}
```

mouseMoveEvent() fonksiyonu fernin hareket ettirilme eylemini halleder. Normalde bu eylemler sadece fare tuğu basılı iken husule getirilirler. Bu davranışı IQWidget::setMouseTracking()[14] fonksiyonunu çağırmak suretiyle değiştirmek mümkündür, ancak bu alıştırma için bizim bu davranışa ihtiyacımız yokru. Sağ veya sol fare tuğuna basmak pikseli temizleyip boyadığı gibi fare tuğuna basılı olduğu halde pikselin üzerinde

14 fareyi takibe al, izle

DRAFT

hareket etmekte onun silinmesine veya boyanmasına neden olur. "İki fare tuğuna aynı anda basılması mümkün olduğundan, QMouseEvent::state() tarafından geri getirilen değer fare tuşunun (hata Shift ve Ctr tuşlarının) "bitwise OR" u dur. Belirli bir tuğun basılmış olduğunu "&" operatörünü kullanarak test ediyoruz ve eğer basılmış ise setImagePixel() fonksiyonunu çağırıyoruz.

```
void IconEditor::setImagePixel(const QPoint &pos, bool opaque)
{
      int i = pos.x() / zoom;
      int j = pos.y() / zoom;
      if (image.rect().contains(i, j))
      {
            if (opaque)
                  image.setPixel(i, j, penColor().rgb());
            else
                  image.setPixel(i, j, qRgba(0, 0, 0, 0));
            QPainter painter(this);
            drawImagePixel(&painter, i, j);
      }
}
```

setImagePixel() fonksiyonu pikseli boyamak veya temizlemek için mousePressEvent() ve mouseMoveEvent() tarafından çağrılır. pos parametresi farenin alet ğzerindeki pozisyonunu verir.

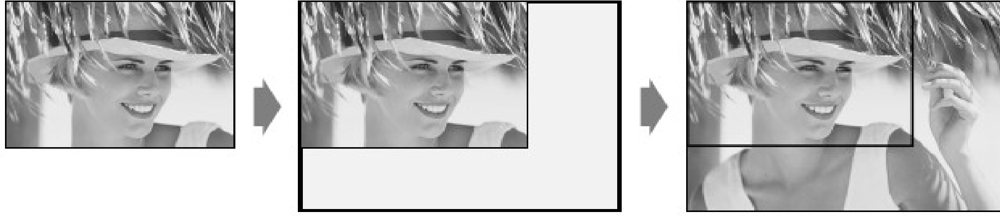
Yapılması gereken ilk iş fare pozisyonunu alet kordinate sisteminden resim kordinate sistemine öevirmek. Bunu yapmak için fare pozisyonunun x ve y kordinatlarını büyütme faktörüne bölmek yeterlidir. Sonra bu noktanın geçerli sınırlar içerisinde olup olmadığını tesbit ediyoruz. Bu işi QImage::rect() ve QRect:: contains() fonksiyonlarını kullanarak kolayca yapabiliriz; burada yapılan iş i değişkeninin 0 ile image.width() - 1 ve j değişkenini ise 0 ile image.height() - 1 arasında olduğunu belirlemektir.

opaque parammetresine bağlı olarak resimdeki pikselleri ya boyuyoruz yada temizliyoruz. Pikseli temizlemek onu şeffaf yapmakla aynı manaya gelir. Sonuçta, drawImagePixel() fonksiyonunu çağırmak suretiyle değişmiş olan her bir pikseli boyuyoruz.

Üye fonksiyonları bitirdik, şimdi yapıcıda kullandığımız WStaticContents seçeneğini ele alacağız. Bu seçenek Qt ye aletin ebatları deüişmesi durumunda içeriğinin değişmemesi gerektiğini ve içeriğinin sol üst köşeye sabitlenmiş olması gerektiğini bildirir. Qt bu durumda aletin ebatının değişmesi durumunda aletin görünmekte olan bölümlerini gerksiz yere boyamaz.

Normalde, bir aletin ebatı değiştiğinde Qt aletin görülebilen bütün alanı için bir boyama etlemi husule getirir. Eğer bir alet WStaticContents seçeneği kullanılarak oluşturulursa bu durumda boyama işlemi sadece daha önce görünmeyen alanlarla sınırlıdır. Eğer bir alet küçültülürse bu durumda hiç bir boyama işlemi gerçekleştirilmez.

Şekil 5.5: WstaticContents seçeneği ile oluşturulan aletin ebatının değiştirilmesi.



IconEditor alrtini böylece tamalamış olduk. Geçmiş bölümlerdeki bilgileri ve misalleri kullanarak IconEditoru başlı başına bir alet gibi kullanan kod yazmak mümkün ki burada bu yeni alet QMainWindow için merkezi alet, bir dizim içerisinde veya QScrollView (p. 145) içerisisnde yer alan çocuk bir alet olabilir. Bir sonraki bölümde bu aletin Qt Designer proğramına nasıl ekleneceğini göreceğiz.

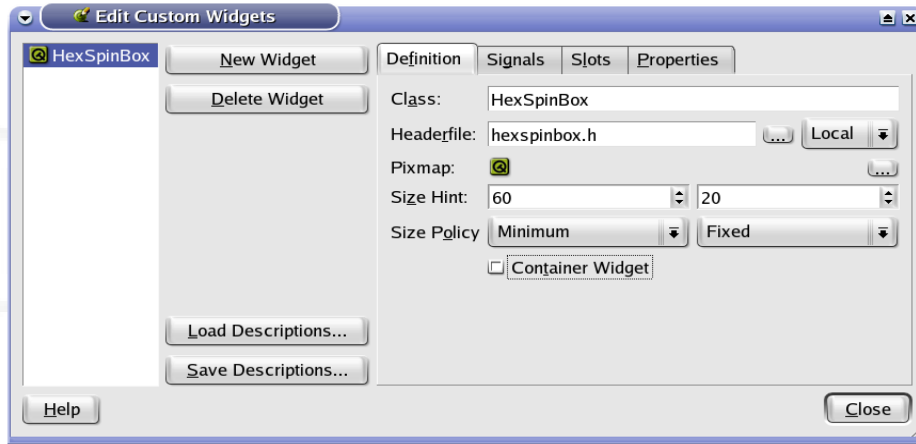
Özel Aletlerin Qt Designer Proğramına Eklenmesi

Özel aletleri Qt Designer içerisinde kullanabilmemiz için önce onu bu aletlerin varlığından haberdar etmeliyiz. Bunu yapmak içim iki metod mevcuttur: basit özel alet yaklaşımı ve "plugin" yaklaşımı. Basit özel alet yaklaşımı Qt Designer altında bir diyaloğa özel alet hakkında bir takım bilgiler girmekten ibarettir. Bu alet daha sonra Qt Designer içinde yapılan formlarla birlikte kullanılabilir ancak alet sadece bir simge ve koyu gri bir dikdörtgen olarak görünür. HexSpinBox aleti şu şekilde Qt Designer proğramına eklenebilir:

1. Tools | Custom | Edit Custom Widget menüsünü seç. Bu Qt Designer in özel alet editörünü açar.
2. New Widget düğmesine tıkla .
3. Sınıfın ismini "MyCustomWidget" yerine "HexSpinBox" ve başlık dosyasını "mycustomwidget.h" yerine "hexspinbox.h" olarak değiştir.
4. Sınıfın ideal boyutunu (size hint) (60, 20) olarak değiştir.
5. Ebat tarzı (size policy) özelliğini (Minimum, Fixed[15]) olarak değiştir.

Bu alete Qt Designer proğramının özel aletler (Custom Widgets) bölümünden ulaşılabilir.

15 sabit



Şekil 5.6: Qt Designer proğraamının özel alet editörü.

"Plugin" yaklaşımı plugin kütüphanesinin oluşturulmasını gerektiriyor ki bu kütüphane Qt Designer tarafından koşma anında yüklenir ve misalleri oluşturulabilir. Bu sayede Qt Designer aletin asıl şeklini form değiştirilirken ve önizleme esnasında kullanır. Bunun nasıl yapılabileceğini göstermek için IconEditor aletini Qt Designer proğramına ekleyeceğiz. Önce QWidgetPlugin sınıfından bir alt sınıf oluşturmalıyız ve sonra bunun bazı hayali fonksiyonlarını yeniden tanımlamalıyız. Bütüm bunlar aynı kaynak dosyasında yapılabilir. Plugin kaynak kodunun iconeditorplugin isimli dizide ve IconEditor aletinin kaynak kodununda aynı seviyedeki iconeditor dizisinde olduğunu varsayıyoruz.

İşte başlık dosyası:

```
#include <qwidgetplugin.h>
#include "../iconeditor/iconeditor.h"
class IconEditorPlugin : public QWidgetPlugin
{
      public: QStringList keys() const;
      QWidget *create(const QString &key, QWidget *parent,
                      const char *name);
      QString includeFile(const QString &key) const;
      QString group(const QString &key) const;
      QIconSet iconSet(const QString &key) const;
      QString toolTip(const QString &key) const;
      QString whatsThis(const QString &key) const;
      bool isContainer(const QString &key) const;
};
```

IconEditorPlugin alt sınıfı bir fabrika sınıf olup IconEditor alrtini kapsar(encapsulates). Üye fonksiyonları eklenmkete olan altin oluşturulması ve onun hakkında bilgi edinilmesi için Qt Designer tarafından kullanılırlar.

```
QStringList IconEditorPlugin::keys() const
{
      return QStringList() << "IconEditor";
}
```

keys()[16] fonksiyonu plugin tarafından tedarik edilen aletlerin listesini tedarik eder. Bu plugin sadece IconEditor aletini ihtiva eder.

```
QWidget *IconEditorPlugin::create(const QString &,
      QWidget *parent,const char *name)
{
      return new IconEditor(parent, name);
}
```

create()[17] fonksiyonu Qt Designer tarafından alet sınınfının bir nesnesini oluşturmak için çağrılır. İlk argüman aletin isöidir. Bu misalde bunu gözardı edebiliriz çünkü bizim misalimizde sadece bir sınıf var. Bütün diğer fonksiyonlarda ilk argümanları olarak sınıfın ismini alırlar.

```
QString IconEditorPlugin::includeFile(const QString &) const
{
      return "iconeditor.h";
}
```

includeFile()[18] fonksiyonu plugin dahilinde olan belirli bir aietin başlık dosyasının ismini verir. Bu başlık dosyası `uic` tarafından husule getirilen koda dahil edilir.

```
bool IconEditorPlugin::isContainer(const QString &) const
{
      return false;
}
```

isContainer()[19] fonksiyonu, söz konusu alet diğer aletleri içerebiln bir alet ise, nüsbet aksi takdirde menfi deüerini üretir. Mesela, QFrame diğer aletleri ihtiva edebilen bir alettir. IconEditor için biz menfi değerini geri gönderiyoruz çünkü bu aletin diğer aletleri ihtiva etmesi saçma olur. Aslında, her alet başka aletleri içine alabilir ancak Qt Designer, isContainer() fonksiyonu menvi değer üreten bir sınıf için, buna izin vermez.

```
QString IconEditorPlugin::group(const QString &) const
{
      return "Plugin Widgets";
}
```

group() fonksiyonu bu aletin mensubu olduğu alet kutusunun ismini verir. Bu isim mevcut değil ise Qt Designer bu alet için yeni bir gurup oluşturur.

```
QIconSet IconEditorPlugin::iconSet(const QString &) const
{
      return QIconSet(QPixmap::fromMimeSource("iconeditor.png"));
}
```

iconSet() fonksiyonu bu özel aleti Qt Designer alet kutusunda temsil edecek olan simgeyi verir.

---

16 anahtarlar

17 oluştur, yap

18 başlık dosyası

19 muhtevi (içeren, içine alan) sınıf

```
QString IconEditorPlugin::toolTip(const QString &) const
{
      return "Icon Editor";
}
```

toolTip() fonksiyonu, farenin Qt Designer alet kutusunda özel aletin üzerinde hareket dolaşıp dururken görüntülenecek olan, alet üpucunu (tooltip) verir.

```
QString IconEditorPlugin::whatsThis(const QString &) const
{
      return "Simge olu turmaya ve de i tirmeye yarayan bir alet";
}
```

whatsThis() fonksiyonu Qt Designer tarafından Bu Ne? (What s This?) için görüntülenecek olan mesajı verir.

```
Q_EXPORT_PLUGIN(IconEditorPlugin)
```

Plugin sınıfının kaynak kodunu ihtiva eden dosyanın sonunda bu plugini Qt Designerin emrine amade edecek olan Q_EXPORT_PLUGIN()[20] makrosunu kullanmamız gerekir:

```
TEMPLATE = lib
CONFIG += plugin
HEADERS = ../iconeditor/iconeditor.h
SOURCES = iconeditorplugin.cpp \
               ../iconeditor/iconeditor.cpp
IMAGES = images/iconeditor.png
DESTDIR = $(QTDIR)/plugins/designer
```

Proje dosyası (.pro) QTDIR isminde Qt nin kurulmuş olduğu diziyi güsteren çevre değişkeninin varlığını varsayar. make yada nmake komutunu icra ettiğinizde plugin derlenip Qt Designer plugin dizisine konur yada kurulur. Plugin kurulduktan sonra, IconEditor aleti Qt Designer da mevcut diğer aletler gibi kullanılabilir.

Çifte Arabellek Kullanımı (Double Buffering)

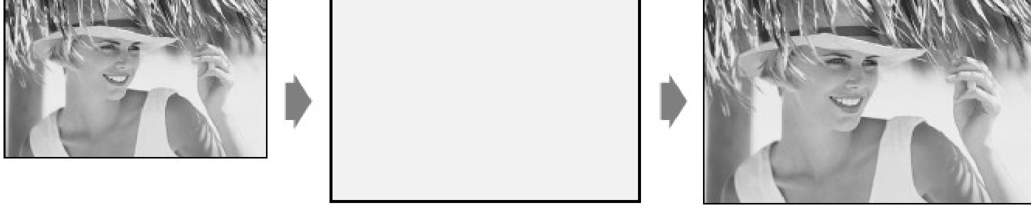
Çifte arabellek kullanımı bir teknik olup daha çevik kullanıcı arabirirmi oluşturmak ve ekrandaki titreşimleri önlemek için kullanılır. Ekrandaki titremeler bir pikselin kısa bir zaman dilimi içerisinde mütaddid renklere boyanmasıyla meydana gelir. Bunun bir tek piksel için meydana gelmesi bir problem teşkil etmezken çok sayıda pikselin etkeilenmiş olması kullanıcının dikkatini bozar veya onu rahatsız eder.

Qt boyama eylemi (paint event) husule getirdiğinde önce aletin renk plaetindeki arkaplan rengini kullanarak aleti temizler. Sonrai, paintEvent() fonksiyonu sadece arkaplan renginden farklı renge sahip olan pikseller boyanır. İki adımdan oluşan bu yaklaşım gayet elverişlidir çünkü biz alet üzerinde arzu ettiğimiz pikselleri boyar geri kalanı hakkında kafa yormayız.

Malesef bu yaklaşım akrandaki titremelerin ana sebebidir. Mesela, kullanıcı aletin boyutunu değiştirdiğinde o alet önce tamamen temizlenir daha sonrada gereklü pikseller yeniden

20 plugini (takılabilir, eklenebilir) ihraç et

boyanır. Titreişm, pencere sisteminin aletin ebatları değiştirilirken içeriğini göstermesi durumunda, çok daha bariz olur çünkü bu esnada alet sğrekli olarak temizlenip boyanır.



Şekil 5.7: Titreşime karşı hiç bir tedbiri olmayan aletin ebatının değiştirilmesi.

IconEditor aletini oluştururken kullanılan `WStaticContents`[21] seçeneği titreşimi önlemek için alınabilecek tedbirlerden birisisdir, ancak bu sadece içerişi ebatına bağlı olmayan aletler için kullanılabilir. Bu tür aletler çok nadirdirler. Çoğu aletler içeriklerini germek suretiyle mevcut olan alanın tamamını kullanmaya çalışırlar. Botutları değiştirildikten sonra tamamaen yeniden boyanmaları gerekir. Bu durumda dahi titreşimi önleyebiliriz ancak çözüm biraz daha complexdir.

Bu doğrultuda ilk kaide aleti oluştururken `WNoAutoErase`[22] seçeneğini kullanmaktır. Bu seçenek Qt ye boya eyleminden çnce aleti silmemesi talimatını verir. Böylece mevcut pikseller aynan kalırken yeni teşhir edilen pikseller tanımlanmamış oluerla.



Şekil 5.8: `WNoAutoErase` seçeneğini kullanan aletin ebatının değiştirilmesi.

WnoAutoErase seçeneği kullanıldığında boyama işlemini halleden fonksiyon bütün pikselleri bizzat boyamalıdır. Boyama eylemi sırasında değiştirilmeyen pikseller önceki renklerini muhafaza ederler ki o her zaman arkaplan rengi ile aynı olmayabilir.

Titreşimi önlemek için uyulması gereken ikinci kaide ise her pikseli yanlız bir defa boyanaktır. Bunu ifa etmenin en kolay yolu aletin tamamını ekranda görünmeyen bir piksel haritasında (pixmap) boyamak ve bu resmi daha sonra bir anda aletin üzerine kopyalamaktır. Bu yaklaşımda piksellerin mğteaddid kere boyanmasında bir mahsur yoktur çünkü resim boyanma esnasında ekranda gösterilmemektedir. İşte buna çifte arabellek adı verilir.

21 sabit, değişmeyen muhteviyat (içerik)
22 otomatik olarak silme

Titreşimi önlemek için özel bir alete çifte arabellek tekniğini eklemek gayet kolaydır. Boyama eylemini halleden fonksiyonun aslının şu şekilde olduğunu varsayalım:

```
void MyWidget::paintEvent(QPaintEvent *)
{
      QPainter painter(this);
      drawMyStuff(&painter);
}
```

Çifte ara bellek kullanan versiyonu şu şekilde olabilir:

```
void MyWidget::paintEvent(QPaintEvent *event)
{
      static QPixmap pixmap;
      QRect rect = event->rect();
      QSize newSize = rect.size().expandedTo(pixmap.size());
      pixmap.resize(newSize);
      pixmap.fill(this, rect.topLeft());
      QPainter painter(&pixmap, this);
      painter.translate(-rect.x(), -rect.y());
      drawMyStuff(&painter);
      bitBlt(this, rect.x(), rect.y(),
             &pixmap, 0, 0, rect.width(), rect.height());
}
```

Önce boyanacak olan alanı[23] içine alacak şekilde, QPixmap in ebatını değiştiriyoruz. QPixmap değişkenin, sürekli bellek ayırma ve silme işleminden kaçınmak için, `static` bir değişken yapıyoruz. Aynı sebepten dolayı Qpixmap in ebatını hiç küçültmüyoruz; QSize::expandedTo() ve QPixmap::resize() fonksiyonlarına yapılan çağrılar pıksel haritasının her zaman yeterşnce büyük olmaını garanti etmek amaçlıdır. Ebat değişikliğinden sonra, piksel haritasını (QPixmap) Qpixmap::fill() fonksiyonunu kullanarak ya aletin silme rengi ile yada arka planına yerleştirilmiş olan resim ile dolduruyoruz. fill() fonksiyonuna verilen ikinci argüman piksel haritasının (QPixmap) sol üst köşesinin alet üzerinde hangi noktaya tekabül ettiğini gösterir. (Bu aletin sabit bir renk ile silinmesi yerine bir resim ile silinmesi durumunda ehemmiyet arzeder.)

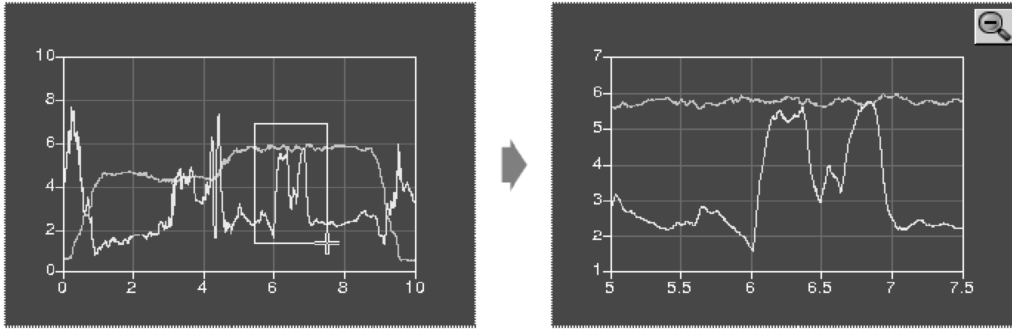
QPixmap sınıf hem QImage hemde QWidget sınıfına benzer. QImage sınıfı gibi resmi muhafaza eder ancak renk derinliği ve mutemelen renk haritası saklı olan bir QWidget gibi ekrana bağlıdır. Eğer pencere sistemi 8-bit modunda çalışıyor ise bütün aletler ve resim haritaları (pixmaps) 256 renkle sınırlandırılır ve Qt otomatik olarak 24-bit renk 8-bit renge çevirir. (Qt nin renk taksim stratejisi Qapplication::setColorSpec() fonksiyonunu çağırmak suretiyle kontrol edilir.)

Sonra, piksel haritası üzerinde deüişiklikler yapabilmek için bir QPainter oluşturuyoruz. `this` müşirini yapıcıya argüman olarak kullanmak suretiyle QPainter in nesnesine bu aletin çzelliklerinden bazılarını (font gibi) kullanması talimatini veriyoruz. QPainter ile boyama işlemini gerçekleştirmeden önce boyacıyı (painter) önce doğtu dikdörtgeni boyaması için

23 alan genelde ya dikdörtgen veya “L” şeklinde olur, ancak bundan çok daha karmaşık olmasıda mümkündür.

kaydırıyoruz. Nihayet piksel haritasını alete kopyalama işlemini kiresel (global) bitBlt()[24] fonksiyonu ile gerçeklerştiriyoruz. Çifte ara bellek, titreşimi önlemek için kullanılmasının yanında bir aletin boyanma işlemi kompleks ise ve sürekli boyama gerektiriyorsa bu durumdada kullanılır. Böyle durumlarda piksel harıtası sürekli olarak alet ile birlikte muhafaza edilir ve boyama eylemine karşılık piksel haritası alete nakledilir. Bilhasse alette küçük bir değişiklik yapılacak ise onun tamamını boyamaya hesaplarına hiç gerek kalmaz.

Bu bölümü Plotter çzel aletini gözden geçirerek bitireceğiz. Bu alet çifte bellek yanında klaviye eylemlerini halledilmesi, kordinat sistemleri ve otomatik olmayan dizim gibi Qt nin diğer yönlerini içermektedir. Plotter aleti kordinatları verilen bir veya daha fazla eğrinin grafiğini çizer. Kullanıcı fareyı kullanarak rsmin üzerine bir dikdörtgen çizdiğide, Plotter bu alanı büyütür. Bu dikdörtgeni çizmek için kullanıcı fare ile resim üzeriğnde bir noktaya tıklar ve fare tuğu basılı olduğu halde fareyi bir başka noktaya taşıyıp tuşu bırakır.



Şekil 5.9: Plotter aletinde bir alanın büyütüşmesi.

Kullanıcı müteaddid kereler dikdörtgen çizmek suretiyle büyütme işlemini tekrarlayabilir. Bu aşamadan sonra Küçült (Zoom Out) ve Büyüt (Zoom In) düğmeleri kullanılarakta küçültme büyütme tekrarlanabilir. Bu düğmeler kullanıcı fareyi kullanarak büyütme yapmadıkça ekranda gözükmezler. Plotter çok sayıda datanın grafiğini ihtiva edebilir. Aynı zmanada bvelli büyütme seviyesine tekabül eden grafik özelliklerini (PlotSettings) ihtiva eden bir yığın muhafaza eder. Şimdi plotter.h dosyasından başlayarak bu sınıfı gözden geçirelim:

```
#ifndef PLOTTER_H
#define PLOTTER_H

#include <qpixmap.h>
#include <qwidget.h>

#include <map>
#include <vector>

class QToolButton; class PlotSettings; typedef std::vector<double>
CurveData;
```

24 bit-block transfer (bit blokunun nakli)

Stabdart <map> ve <vector> başlık dosyalarınıda dahil ediyoruz. Standart (std) isim alanındakı (namespace) tanımlanmış bütün sembolleri, tavsiye edilmediğnden, umumi isim alanına dahil etmiyoruz. Grafiğn datasını (CurveData) std::vector<double> türündebir değişken olarak tanımlıyoruz. Grafiği oluşturan noktaların datalarını x ve y çifti olarak bu vektörde tutacağız. Mesela, (0, 24), (1, 44), (2, 89) noktalarından oluşan bir grafik [0, 24, 1, 44, 2, 89] vektörü şeklinde muhafaza edilir.

```
class Plotter : public QWidget
{
      Q_OBJECT
public:
      Plotter(QWidget *parent = 0, const char *name = 0,
                  WFlags flags = 0);
      void setPlotSettings(const PlotSettings &settings);
      void setCurveData(int id, const CurveData &data);
      void clearCurve(int id);
      QSize minimumSizeHint() const;
      QSize sizeHint() const;
public slots:
      void zoomIn();
      void zoomOut();
```

Grafiği oluşturmak için iki tane umumi (public) fonksiyon ve büyütme ve küçültme işlemleri için ise iki tane yine umumi dilim tedarik ediyoruz. Bunlara ilaveten QWidget sınıfından miras kalan minimumSizeHint() ve sizeHint() fonksiyonlarınıda yeniden tanımlıyoruz.

```
protected:
      void paintEvent(QPaintEvent *event);
      void resizeEvent(QResizeEvent *event);
      void mousePressEvent(QMouseEvent *event);
      void mouseMoveEvent(QMouseEvent *event);
      void mouseReleaseEvent(QMouseEvent *event);
      void keyPressEvent(QKeyEvent *event);
      void wheelEvent(QWheelEvent *event);
```

Sınıfnı mahfuz (protected) bölümünde yenidan tanımlanması gereken QWidget sınınfının eylem hallledicileri tanımlıyoruz.

```
private:
      void updateRubberBandRegion();
      void refreshPixmap();
      void drawGrid(QPainter *painter)c
      void drawCurves(QPainter *painter);

      enum { Margin = 40 };

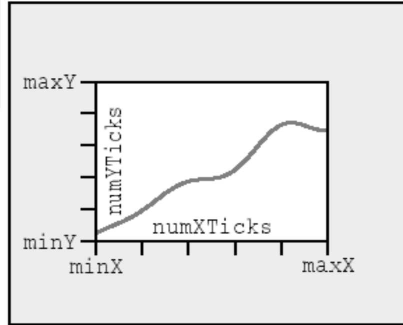
      QToolButton *zoomInButton;
      QToolButton *zoomOutButton;
      std::map<int, CurveData> curveMap;
      std::vector<PlotSettings> zoomStack;
      int curZoom;
      bool rubberBandIsShown;
      QRect rubberBandRect;
      QPixmap pixmap;
```

```
};
```

Sınıfın husus (private) bölümünde bir sabit (constant) , aletin boyanması işlemini yerine getirmek için bir kaç fonksiyon ve üye değişkenleri tanımladık. Margin[25] sabiti grafiğin etrafında boşluk bırakmak için kullanılır. Üye değişkenlerinden birirsi Qpixmap türünden pizmap isimli değişkendir. Bu değişken aşetin üzerine çizilen resmin[26] bir kopyaını tutar ki bu ekranda görünenin aynıdır. Grafik daima önce akranda görünmeyen piksel haritası üzerine çizilir; sonra bu resim alet üzerine trandfer edilir yada kopyalanır.

```
class PlotSettings
{
public:
      PlotSettings();
      void scroll(int dx, int dy);
      void adjust();
      double spanX() const { return maxX - minX; }
      double spanY() const { return maxY - minY; }
      double minX;
      double maxX;
      int numXTicks;
      double minY;
      double maxY;
      int numYTicks;
private:
      void adjustAxis(double &min, double &max, int &numTicks);
};
#endif
```

PlotSettings sınıfı x ve y eksenlerinin sınırlarını ve bu eksenler üzerindeki ticks lerin sayılarını belirler. Şekil 5.10 da PlotSettings nesnesi ile ile Plotter aletinin ölçeği (scale) arasındaki ilişki gösterilmektedir. Adeten numXTicks ve numYTicks değerleri ile grafik üzerinde gösterilen küçük çizgilerin sayısı farklıdır; numXTicks değişkeninin değeri 5 ise, Plotter x ekeseni üzerinde altı tane küçük çizgi çizer. Bu daha sonra yapılacak olan hesaplamalarda kolaylık sağlar.



Şekil 5.10: PlotSettings sınıfının üye değişkenleri.

Şimdi bu sınfın kaynak koduna bakalım:

25 haşiye, marjin

26 Alet üzerine çizimiş bir resim/imaj olsun yada olmasın, burada resimden maksat aletin görünümüdür.

```
#include <qpainter.h>
#include <qstyle.h>
#include <qtoolbutton.h>

#include <cmath>
using namespace std;

#include "plotter.h"
```

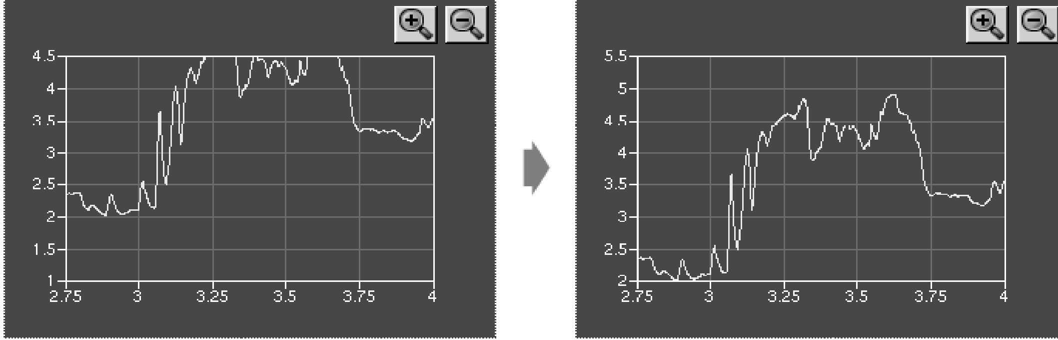
Gerekli başlık dosyalarını ekledik ve standart isim alanından bütün sembolleri umumi (global) isim alanına dahil ettik.

```
Plotter::Plotter(QWidget *parent, const char *name, WFlags flags)
      :QWidget(parent, name, flags | WNoAutoErase)
{
      setBackgroundMode(PaletteDark);
      setSizePolicy(QSizePolicy::Expanding,
                  QSizePolicy::Expanding);
      setFocusPolicy(StrongFocus);
      rubberBandIsShown = false;
      zoomInButton = new QToolButton(this);
      zoomInButton->setIconSet(QPixmap::fromMimeSource
            ("zoomin.png"));
      zoomInButton->adjustSize();
      connect(zoomInButton, SIGNAL(clicked()),
            this, SLOT(zoomIn()));
      zoomOutButton = new QToolButton(this);
      zoomOutButton->setIconSet( Qpixmap::fromMimeSource
            ("zoomout.png"));
      zoomOutButton->adjustSize();
      connect(zoomOutButton, SIGNAL(clicked()),
            this, SLOT(zoomOut()));
      setPlotSettings(PlotSettings());
}
```

Plotter sınıfının ata (parent) ve isim (name) parametrelerine ilaveten birde `flags` parametresi vardır. Bu parametre üst sınıfın, yani QWidget sınıfının, yapıcısna `WNoAutoErase` ile birlikte gönderilir. Bu parametre bilhassa alet başlıbaşına bir pencere olarak kullanılacak ise çok faydalıdır çünkü kullanıcının pencerenın başlığını ve çerçevesini değiştirmesine izin verir.

setBackgroundMode() fonksiyonuna yapılan çağrı QWidget için aletin silinmesi içim arka plan rengi yerine koyu paletin koyu renk bölümünün kullanılması talımatını verır. Her nekadar WNoAutoErase seçeneğini kullanmış olsakta paintEvent() fonksiyonun boyama fırsatı elde etmeinden önce aletin büyültülmesi esnasında tezahür eden piksellerin Qt tarafından boyanması için bir renge ihtiyaç vardır. Plotter aletinin arka plan rengi koyu olacağindan aletin büyültülmesi ile açığa çıkan piksellerin koyu renge boyanması gayet müsaittir. setSizePolicy() fonksiyonunu çağırmakla aletin ebat değiştirme tercihini her iki yön için QSizePolicy::Expanding olarak ayarlıyoruz. Bu aleti içeren dizim mekanizmasına aletin büyümeyü tercih ettiğini ancak gerektiğinde büzülebileceğini bildirir. Bu ayar ekramda çok yer kaplayacak olan aletler için normaldir. Aletlerin normal (default) ebat değiştirme tercihleri QSizePolicy::Preferred olarak ayarlanmıştır kı bu aletin ideal boyutunu

trcih ettiğini ancak gerektiğinde minimum boyutuna küçülebileceği gibi sınırısz olarakta büyüyebilir. setFocusPolicy() fonksiyonun yapılan çağrı alete ya tab tuğuna basılarak yada fare tıklamasıyla odak noktası olma yeteneğini verir. Plotter aleti odak noktası olduğunda klavıyeden tuş komutları ile idare edilebilir. Plotter aleti bir kaç tuş komutunu anlamaktadır: + büyütmrk için, - küçültmek için, ve oklar ise sağa, sola, aşağı ve yukarı doğru kaydırmalar için kullanılabilir.



Şekil 5.11: Plotter aletinin kaydırılması.

Yapıcı içerisinde simgeleri ile birlikte iki tane alet düğmesi (QtoolButtons) oluşturuyoruz. Bu düğmeleri kullanıcı büyütme/küçültme işlemleri için kullanır. Bu düğmelerin simgeleri bir imaj topluluğunda muhafaza edilmektdir. Plotter aletini kullanan herhangi bir proğram proje dosyasına (.pro) su satırları eklemek zorundadır:

```
IMAGES += images/zoomin.png \
images/zoomout.png
```

adjustSize() fonksiyonu yapılan çağrı basıtası ile düğmelerin büyüklükleri ideal büyüklüklrine ayarlarnır. En sonda setPlotSettings() fonksiyonu yapılan çağrı geri kalan ayarlamalrı tamamalr.

```
void Plotter::setPlotSettings(const PlotSettings &settings)
{
      zoomStack.resize(1);
      zoomStack[0] = settings;
      curZoom = 0;
      zoomInButton->hide();
      zoomOutButton->hide();
      refreshPixmap();
}
```

setPlotSettings() fonksiyonu grafiğin görüntülenmesi için kullanılacak olan grafik ayarlarını (PlotSettings) düzenler. Bu fonksiyon Plotter ın yapıcısı tarafından çağrılır ve bu sınıfın kullanıcısı tarafındanda çağrılabilir. Grafik aleti (plotter) ilk başladığında normal büyüklük ile başlar. Kullanıcı her dafasında büyütme tuşuna basdığında PlotSettings in yeni bir timsali oluşturulur ve büyütme yığınına konur. Büyütme yığını iki üye değişken tarafından temsil edilir:

- zoomStack muhtelif büyütmr ayarlarını vector<PlotSettings> türünde bir vektörde tutar.

- curZoom zoomStack daki aktif olan PlotSettings in indeksini tutar.

setPlotSettings() fonksiyonunun çağrılmasının ardından büyütme yığınında sadece bir girdi mevcuttur ve Büyüt (Zoom In) ve Küçült (Zoom Out) düğmeleri saklıdırlar. Bu düğmeler zoomIn() ve zoomOut() dilimlerinden show() fonksiyonunu çağırıncaya kadar saklı klırlar. (Normalde bir alet için show() fonksiyonu çağrıldığında onun bütün çocuklarıda görüntülenir. Ancak çocuklardan herhangi birisi bizzat hide() fonksiyonu çağrılıp saklanmışsa onu görüntülemek için yine show() fonksiyonunu çağırmak gerekir. Bu çocuğun atasına yapılan show() çağrısı çocuğu görüntülemez.) Ekranı güncelleştirmek için refreshPixmap() fonksiyonun çağrılması elzemdir. Genelde, update() fonksiyonu çağrılır, ancak biz burada biraz farklı çaşığıyoruz çünkü biz her an QPixmap i güncel tutmak istiyoruz. Piksel haritasının (pixmap) güncelleştirilmesindn sonra refreshPixmap() fonksiyonu update() fonksiyonunu çağırır ve piksel haritasını aletin üzerine kopyalr.

```
void Plotter::zoomOut()
{
      if (curZoom > 0)
      {
            --curZoom;
            zoomOutButton->setEnabled(curZoom > 0);
            zoomInButton->setEnabled(true);
            zoomInButton->show();
            refreshPixmap();
      }
}
```

zoomOut() dilimi büyültülmüş olan grafiği küçültür. Hali hazırdaki büyütme seviyesini düşürür ve eğer grafik daha fazla küçültülebilecek durumda ise Küçült düğmesini muktedir (enabled) yapar. Büyüt düğmesi muktedir yapılır ve görüntülenir aynı zamanda refreshPixmap() fonksiyonun çağrılması ile ekram güncelleştirilir.

```
void Plotter::zoomIn()
{
      if (curZoom < (int)zoomStack.size() - 1)
      {
            ++curZoom;
            zoomInButton->setEnabled( curZoom < (int)
                  zoomStack.size() - 1);
            zoomOutButton->setEnabled(true);
            zoomOutButton->show();
            refreshPixmap();
      }
}
```

Kullanıcı daha önce grafiği büyütmüş ve daha sonrada küçültmüş ise bir sonraki büyütme seviyesine ait grafik ayarları (PlotSettings) büyütme yıüınında yer alır ve istersek grafiği büyütebiliriz. (Aksi takdirde büyütme işlemini grafiğin üzerine dikdörtgen çizmek suretiyle büyütme yapmamız gerekir.) Bu dilim curZoom değişkenini artırır ve büyütme yığınında daha derine inilir aynı zamanda Büyüt (Zoom In) düğmesini, daha fazla büyütmek mümkün ise, muktedir yada malul yapar ve Küçült düğmesini muktedir kılar ve gösterir. Yine refreshPixmap() fonksiyonunu çağırmak suretiyle grafiğin en son ayarları kullanmasını

sağlıyoruz.

```
void Plotter::setCurveData(int id, const CurveData &data)
{
      curveMap[id] = data;
      refreshPixmap();
}
```

setCurveData() fonksiyonu belirli bir kimliğe (ID) sahip eğrinin datasını blirler. Eğer bu kimlikte bir eğri grafikte mevcut ise eski eğri yeni data ile değiştirilir, aksi takdirde yeni data/eğri grafiğe eklenir. Eğriler map<int, CurveData> türündeki curveMap üye değişkeninde tutulurlar. Sonra update() yerine ekranı güncelleştirmek için refreshPixmap() fonksiyonunu çaşırıyoruz.

```
void Plotter::clearCurve(int id)
{
      curveMap.erase(id);
      refreshPixmap();
}
```

clearCurve() fonksiyonu eğriyi curveMap değişkeninden siler.

```
QSize Plotter::minimumSizeHint() const
{
      return QSize(4 * Margin, 4 * Margin);
}
```

minimumSizeHint() fonksiyonu sizeHint() fonksiyonuna benzer; sizeHint() bir aletin ideal boyutunu verdiği gibi minimumSizeHint() da bir aletin en makul minimum ebatını verir. Bir dizim mekanizması aleti hiç bir zaman minimum ebatının altında olacak şekilde boyutlandırmaz. Biz 160 × 160 değerini geri gönderiyoruzki bu kenarlada haşiye yani marjin için yer ve grafiğin kendisi için kullanılacak yerdir. Bu ebatın altında alet işe yaramaz hale gelir.

```
QSize Plotter::sizeHint() const
{
      return QSize(8 * Margin, 6 * Margin);
}
```

sizeHint() fonksiyonunda aletin ideal ebatını marjin büyüklüğüne oranla hesaplıyoruz ve genişliğinin yüksekliğine oranın ise 4:3 olarak blirliyoruz.

Böylelikle Plotter sınıfının umumi (public) fonksiyon ve dilimlerini gözden geçirmiş oluyoruz. Şimdi eylem halledicilerine bakalım.

```
void Plotter::paintEvent(QPaintEvent *event)
{
      QMemArray<QRect> rects = event->region().rects();

      for (int i = 0; i < (int)rects.size(); ++i)
            bitBlt(this, rects[i].topLeft(), &pixmap, rects[i]);

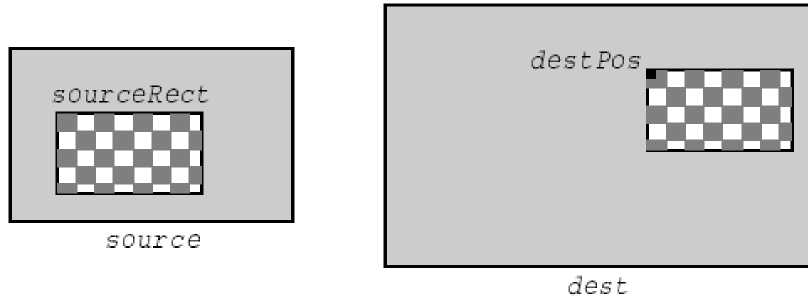
      QPainter painter(this);
```

```
        if (rubberBandIsShown)
        {
            painter.setPen(colorGroup().light());
            painter.drawRect(rubberBandRect.normalize());
        }
        if (hasFocus())
        {
            style().drawPrimitive(QStyle::PE_FocusRect, &painter,
                rect(), colorGroup(),
                QStyle::Style_FocusAtBorder,
                colorGroup().dark());
        }
    }
```

Normalde bütün çizimler genelde paintEvent() fonksiyonunda gerçekleştirilirler. Ancak burada bütün çizim yada boyama işlemi daha önce refreshPixmap() fonksiyonunda gerçekleştirildiğinden yapılacak tek şey piksel harıtasını aletin üzerine kopyalamaktır. QRegion::rect() fonksiyonu boyanması gereken dikdörtgenlerin listesini (QRects) verir. bitBlt() fonksiyonunu kullanarak bu dikdçrtgenleri piksel haritasından aletin üzerine kopyalıyoruz. bitBlt() umumi (global) fonksiyonunun nehvi yani sintaksı şu şekildedir:

bitBlt(dest, destPos, source, sourceRect);

ki burada source (kaynak) nesnesi ya piksel haritasını yada kaynak aleti verir, sourceRect kaynaktaki kopyalanacak dikdörtgeni verir, dest ise hedefi verirki burada hedef ya bir piksel haritası yada bir diğer alettir ve nihayet destPos ise hedefteki üst sol köşenin pozisyonunu verir.



Şekil 5.13: Rastgele bir dikdörtgenin aletten piksel haritasına ve piksel haritasından alete kopyalanması[27].

Alanın tamamı için bitBlt() fonksiyonunu yanlız bir defa çağırmakta mümkün. Ancak, fare eylemlerini halleden fonksiyonlarda sürekli olarak update() fonksiyonunu çağırıp fare ile çizilen büyütme dikdörtgenini sürekli olarak (birazdan göreceğimiz gibi) silip çizdiğimizden ve büyütme dikdörtgeni aslında dört tane küçücük dikdörtgenden ibaret (iki tane bir piksel genişliğinde ve iki tanede bir piksel yüksekliğinde) olduğundan , bitBlt() fonksiyonunu tüm lan için çağırmak yerine bu ufacık dikdörtgenlerin her biri için çağırmamız bize zaman

27 `source`, kaynak ve `dest` (destination) ise varılacak yer demektir.

kazandırıyor. Grafik ekranda görüntülendiğinde büyütme dikdörtgenini üzerine çizip odak dikdörtgeninide onun üzerine çiziyoruz. Büyütme dikdörtgenini aletin paletindeki açık renkleri kullanarak çiziyoruz taki koyu arka plan üzerinde görülebilsin. Dikkat edilmelidirki biz çizimi direk olarak aletin üzerine yapıyoruz böylece akranda gösterilmeyen aletin piksel haritasına dokunmamış oluyoruz. Odak dikdörtgeni aletin stillerinin (styles) drawPrimitive() fonksiyonunu kullanarak çizilirki bu fonksiyonun ilk argümanı `PE_FocusRect` dır. QWidget::style() fonksiyonu aletin stilini verir kı buda aletin çiziminde kullanılır. Qt de bir aletin stili QStyle sınıfının bir alt sınıfıdır. Qt ile hazır gelen stiller şunlardır: QWindowsStyle, QWindowsXPStyle, QMotifStyle, ve QMacStyle. Bu stillerden her biri QStyle sınıfının hayali fonksiyonlarını yeniden tanımlamak suretiyle taklit etmeye çalıştıkları stili çizerler. drawPrimitive() fonksiyonu bunladan biridir; o iptidai yani temel elementlerden olan panellerö düğmeler ve odak dikdörtgenleri gibi elemanların çizimini yapar. Alet sitili genelde bir proğram içerisindeki (Qapplication::style()) bütün aletler için aynıdır , ancak arzu edilirse her bir aletin stili Qwidget::setStyle() fonksiyonunu kullanmak suretiyle değiştirilebilir. QStyle sınıfından alt sınıf oluşturmak suretiyle yeni ve özel bir stil oluşturmak mümkündür. Bu proğrama veya proğramlar gurubuna hususi stil vermek için yapılr. Her ne kadar hali hazırdaki işletim sisteminin stilini kullanmak tavsiye edilir isede, macerayı seven proğramcılar için Qt gayet elverişlidir. Qt ile gelen aletlerin tamamı nerde kendilerini boyamak için nerede ise tamamen QStyle sınıfına dayanırlar. Bu yüzden onlar Qt trafından desteklenen işletim sistemlerinin herbirinde sanki o sistemin kendi aletleri imiş gibi görünürler. Özel aletler ya QStyle sınıfını kullanarak kendilerini boyamak yada Qt ye ait aletleri çocul alet olarak kullanmak suretiyle kendilerini değişik sitillere duyarlı yapabilirler. Plotter aleti için biz her iki yaklaşımıda kullanıyoruz: Odak dikdörtgeni QStyle sınıfını kullanarak çizilmiştir ve Büyüt (Zoom In) ile Küçült (Zoom Out) düğmeleri Qt nin orjinal aletlerindendir.

```
void Plotter::resizeEvent(QResizeEvent *)
{
    int x = width() - (zoomInButton->width() +
                       zoomOutButton->width() + 10);
    zoomInButton->move(x, 5);
    zoomOutButton->move(x + zoomInButton->width() + 5, 5);
    refreshPixmap();
}
```

Plotter aletinin ebatı değiştirildiğinde Qt ebat değiştirildi eylemi (resize event) husule getirir. Burada biz resizeEvent() fonksiyonununu yeniden tanımladık taki Büyüt ve Küöült düğmelerini Plotter aletinin sağ üst köşesine yerleştirelim. Büyüt ve Küçült düğmelerini yanyana gelecek şekilde aralarında 5 piksellik boşluk ve ataları olan aletin üst kenarı ile sağ kenarı taraflarındada yine 5 piksellik kalacak şekilde yerleştiriyoruz. Eğer düğmelerin aletin sol üst köşesine (bu noktanın kordinarı (0,0) dır) çakılı kalmalarını arzu etseydik bu düğmeleri Plotter aletinin yapıcısında o noktaya taşırdık. Ancak biz üst sağ köşeyi takip etmek istiyoruz ki bu nokyanın kordinatı aletin büyüklüğüne bağlıdır. Bundan dolayı resizeEvent() fonksiyonunu yeniden tamamlayıp pozisyonu burada belirledik. Plotter sınıfının yapıcısında düğmeler için herhangi bir pozisyon belirlemesi yapmadık. Bu bir

problem teşkil etmez çünkü Qt bir aleti ilk defa görüntülemeden önce ebat değişti eylemi (resize event) husuşe getirir. resizeEvent() fonksiyonunu yeniden tanımlayıp aletleri bizzat el ile yerleştirmek yerine bir dizim mekanizması (QGridLayout gibi) kullanıp çocuk aletlerin yerleştirmeside alternatif olarak kullanılabilirdi. Malesef bu yaklaşım biraz daha zor olnakla birlikte dha fazla kaynak kullanırdı. Bir aleti sıfırdan, bizim burada yaptığımız gibi, oluştururken genelde doğru yaklaşım çocuk aletleri el ile yerleştirmektir. Eylem halledicisinin sonunda refreshPixmap() fonksiyonunu çapırarak piksel haritasını teni ebatta çiziyoruz.

```
void Plotter::mousePressEvent(QMouseEvent *event)
{
      if (event->button() == LeftButton)
      {
            rubberBandIsShown = true;
            rubberBandRect.setTopLeft(event->pos());
            rubberBandRect.setBottomRight(event->pos());
            updateRubberBandRegion();
            setCursor(crossCursor);
      }
}
```

Kullanıcı sol fare tuşuna bastığında büyütme dikdörtgeninini görüntğlemeye başlıyoruz. Bu `rubberBandIsShown`[28] değişkeninin değerini müsbet yapmak, `rubberBandRect`[29] üye değişkenine hali hazırdaki fare pozisyonunu yerleştirmek, büyütme dikdörtgeninin boyanması için gerekli olan boyama eylemine randevğ almak, ve fare imlecini ok yerine artı şeklinde değiştirmekten ibarettir.

Qt fare imlecinin şeklini kontrol etmek için iki mekanizma tedarik eder:

- Qwidget::setCursor() fonksiyonu fare alet üzerinde seyrederken imlecin hangi şekil alacağını belirler. Bir alet için fare imlecinin şeklş belirlenmemiş ise o atasının imlecinin şeklini kullanır. Normalde üst seviyedeki bir alet için ok şeklindeki imleç kullanılır.
- QApplication::setOverrideCursor() fonksiyonu imlecin şeklini bütün bir proğram için değiştirir. Bu fonksiyonununbelirlediği şekil her bir aletin kendine has şeklinden daha önceliklidir takı restoreOverrideCursor() çağrılıp imleç normal şekline geri çevrilsisn.

Dördüncü bçlümde QApplication::setOverrideCursor() fonksiyonunu `waitCursor`[30] seçeneği ile çağırmak suretiyle proğramın imlecini standart bekleme imlecine çevirdik.

```
void Plotter::mouseMoveEvent(QMouseEvent *event)
{
      if (event->state() & LeftButton)
      {
            updateRubberBandRegion();
            rubberBandRect.setBottomRight(event->pos());
            updateRubberBandRegion();
```

28 büyütme dikdörtgeni gösteriliyor

29 büyütme dikdörtgeni

30 bekleme imleci

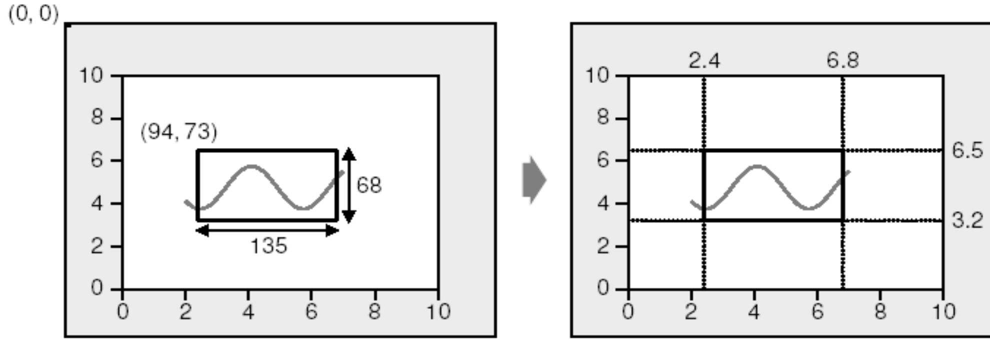
```
        }
}
```

Kullanıcı sol fare tuşunu basılı tuttuğu halde fareyi oynatırsa updateRubberBandRegion() fonksiyonunu çağırarak büyütme dikdörtgeninin bulunduğu yeri yeniden boyamak için bir boyama eylemine randevü alıyoruz, farenin yerdeğiştirmiş olmasını hesaba katmak için `rubberBandRect` değişkenini güncelleştiriyoruz ve ikinci kez updateRubberBandRegion() çağırmak suretiyle büyütme dikdörtgeninin vardığı alanı yeniden boyuyoruz. Aslında bu büyütme dikdörtgenini siler ve yeni yerinde çizer. rubberBandRect değişkenin QRect türündendir. Bir QRect ya (x, y, w, h) şeklinde (burada (x, y) dikdörtgenin sol üst köşesinin pozisyonununu ve w×h ise genişliğini ve yüksekliğini verir) yada dikdörtgenin sl üst ve sağ alt köiesinin kordinatlarını vermek suretiyke oluşturulabilir. Burada biz ikinci yaklaşımı kullandık. Şöyleki kullanıcının fareyi ilk tıkladığı yer dikdörtgenin sol üst köşesini ve farenin hali hazırdaki pozisyonu ise dikdörtgenin sağ alt köşesini verir. Kullanıcı fareyi ilk tıklamadan sonra yukarı veya sola döğru hareket ettirirse bu durumda büyütme dikdörtgenin sağ alt köşesi diya addettiğimiz köşesi sol üst kişe olarak ittihaz ettiğimiz köşenin solunda veya üzerinde yer alabilir. Bu durumda dikdörtgenin (QRect) ya genişliği yada yüksekliği negatif olur. QRect sınıfının normalize() diye bir fonksiyonu mevcuutrki bu fonksiyon negatif genişlik veya yükseklik ortaya çıkmaması için köşelerin kordinatlarında gerekli ayarlamaları yapar.

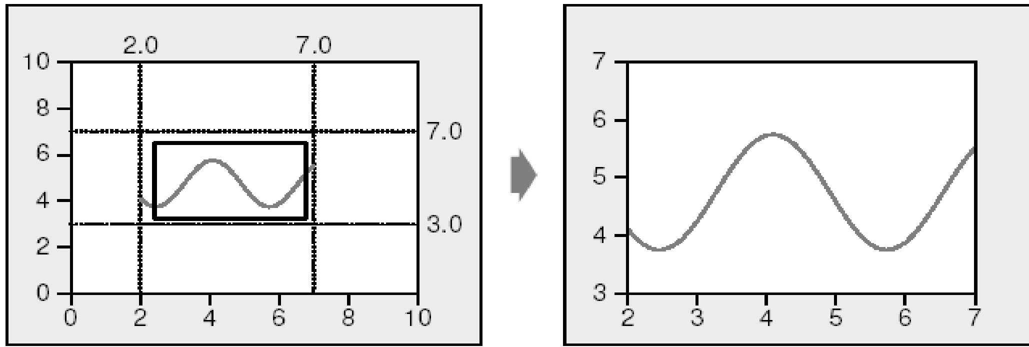
```
void Plotter::mouseReleaseEvent(QMouseEvent *event)
{
      if (event->button() == LeftButton)
      {
            rubberBandIsShown = false;
            updateRubberBandRegion();
            unsetCursor();
            QRect rect = rubberBandRect.normalize();
            if (rect.width() < 4 || rect.height() < 4)
                  return;
            rect.moveBy(-Margin, -Margin);
            PlotSettings prevSettings = zoomStack[curZoom];
            PlotSettings settings;
            double dx = prevSettings.spanX() / (width() - 2 *
                                  Margin);
            double dy = prevSettings.spanY() / (height() - 2 *
                                  Margin);
            settings.minX = prevSettings.minX +
                                  dx * rect.left();
            settings.maxX = prevSettings.minX +
                                  dx * rect.right();
            settings.minY = prevSettings.maxY -
                                  dy * rect.bottom();
            settings.maxY = prevSettings.maxY - dy * rect.top();
            settings.adjust(); zoomStack.resize(curZoom + 1);
            zoomStack.push_back(settings);
            zoomIn();
      }
}
```

Kullanıcı sol fare tuşunu bıraktığında, büyütme dikdörtgenini silip imleci eski normal haline çeviriyoruz. Büyütme dikdörtgeni en az 4x4 büyüklüğünde ise büyütme işlemini gerçekleştiriyoruz. Eğer büyütme dikdörtgeni 4x4 den daha küçük ise kullanıcı muhtemelen yanlışlıkla alete tıkladı yada aleti odak noktası yapmak için onun üzeribe tıkladı varsayarak hiç birşey yapmıyoruz.

Büytme işlvini görecek olan kod biraz karmaşık. Bunun sebebi iki kordinat sistemi ile birden çalışmamızdır: aletin kordinatları ve grafik (plotter) kordinatları. Burada yapılan işin çoğu büyütme dikdörtgenini (`rubberBandRect`) alet kordinatından grafik kordinatına çevirmektir. Bu çeviri yapıldıktan sonra PlotSettings::adjust() fonksiyonu çağırıp değerleri yuvarladıktan sonra makul olan küçük çizgilerin sayılarını buluyoruz.



Şekil 5.14: Büyütme dikdörtgeninin alet kordinat sisteminden çizici (plotter) kordinat sistemine çevrilmesi.



Şekil 5.15: Çizici kordinatlarını ayarlayıp büyütme dikdörtgeninin içindeki alanı büyütmek.

Sonra büyütme işlemini gerçekleştiriyoruz. Büyütme işlemini gerçekleştirmek için daha yeni hasaplamiş olduğumuz grafik ayarlarını (PlotSettings) büyütme yığınının üstüne ekleyip asıl büyütme işlemini yapacak olan zoomIn() fonksiyonunu çağırıyoruz.

```
void Plotter::keyPressEvent(QKeyEvent *event)
{
      switch (event->key()){
      case Key_Plus:
            zoomIn();
            break;
      case Key_Minus:
```

```
                zoomOut();
                break;
            case Key_Left:
                zoomStack[curZoom].scroll(-1, 0);
                refreshPixmap();
                break;
            case Key_Right:
                zoomStack[curZoom].scroll(+1, 0);
                refreshPixmap();
                break;
            case Key_Down:
                zoomStack[curZoom].scroll(0, -1);
                refreshPixmap();
                break;
            case Key_Up:
                zoomStack[curZoom].scroll(0, +1);
                refreshPixmap();
                break;
            default:
                QWidget::keyPressEvent(event);
            }
    }
```

Plotter aleti odak noktası iken kullanıcı bir tuşa bastığında keyPressEvent() fonksiyonu çağrılır. Bu fonksiyonu yeniden tanımladık ki şu tuşlara basıldığında işlem yapsın:+, -, Yukarı (Up), Aşağı (Down), Sol (Left) ve Sağ (Right). Eğer kullanıcı bu altı tuş haricinde bir tuşa basarsa o zaman üst sınıfın keyPressEvent() fonksiyonunu çağırıyoruz. Kolaylık olması için Shift, Ctrl, ve Alt tuşlarını kullanmadık ki bunlar QkeyEvent::state() fonksiyonu tarafından tedarik edilirler.

```
void Plotter::wheelEvent(QWheelEvent *event)
{
    int numDegrees = event->delta() / 8;
    int numTicks = numDegrees / 15;
    if (event->orientation() == Horizontal)
        zoomStack[curZoom].scroll(numTicks, 0);
    else
        zoomStack[curZoom].scroll(0, numTicks);
    refreshPixmap();
}
```

Teker eylemleri (wheel events) farenin tekeri çevrilince meydana gelir. Çoğu farelerin bir düşey tekeri vardır ancak bazıları aynı zamanda yatay tekerde tedarik ederler. Qt her iki tür tekeri desteklemektedir. Teker eylemleri o an odak noktası olan alete giderler. delta() fonksiyonu tekerin ne kadar çevrildiğini derecenin sekizde biri olarak verir. Fareler genelde 15 derecelik adımlarla çalışırlar.

Farenin tekeri en yaygın olarak kaydırma çubuğunu hareket ettirmek için kullanılır. QScrollView sınıfından kaydırma çubukları tedarik etmek için alt sınıf oluşturduğumuzda (6. Bölümde ele alınacaktır) QScrollView teker eylemlerini otomatik olarak halleder bundan dolayı burada wheelEvent() fonksiyonunu yeniden tanımlamamıza gerek yoktur. QListView, QTable, ve QTextEdit gibi QScrollView sınıfının varisleri olan Qt sınıfları teker eylemlerini

hiç ilave koda gerek kalmadan desteklerler.

Böylece eylem halledicilerin kodlamasını bitirmiş oluyoruz. Şimdi hususi (private) fonksiyonları gözden geçirelim.

```
void Plotter::updateRubberBandRegion()
{
      QRect rect = rubberBandRect.normalize();
      update(rect.left(), rect.top(), rect.width(), 1);
      update(rect.left(), rect.top(), 1, rect.height());
      update(rect.left(), rect.bottom(), rect.width(), 1);
      update(rect.right(), rect.top(), 1, rect.height());
}
```

updateRubberBand()[31] fonksiyonu mousePressEvent(), mouseMoveEvent() ve mouseReleaseEvent() fonksiyonları tarafından büyütme dikdörtgenini silip yeniden çizmesi için çağrılır. Bu fonksiyon update() fonksiyonun yapılan dört çağrıdan ibarettir ki bunlar büyütme dikdörtgeninin kapladığı dört küçük dikdörtgeni çizmek için gerekli olan boyama eylemine randevü alırlar.

31 büyütme dikdörtgenini güncelleştir

NOT Kullanarak Büyütme Dikdörtgeninin Çizilmesi

Büyütme dikdörtgeni çizmek için yaygın olarakan kullanılan bir metod ise NOT (veya XOR) matematiksel işlemciyi kullanmaktır ki bu büyütme dikdörtgenindeki her bir pikseli onun zıddı bir bit nakışı ile değiştirir. İşye updateRubberBandRegion() fonksiyonunun bu tekniği kullanarak yazılmış yeni versiyonu:

```
void Plotter::updateRubberBandRegion()
{
      QPainter painter(this);
      painter.setRasterOp(NotROP);
      painter.drawRect(rubberBandRect.normalize());
}
```

setRasterOp() fonksiyonu boyacının (painter) raster operasyonunu NotROP olarak ayarlar. Orjinal versiyonunda varsayılan (default) değer olan CopyROP kullandık ki bu boyacıya (QPainter) yeni değeri orşinalinim üstüne kopyalaması talimatını verir. updateRubberBandRegion() fonksiyounu aynı kordinatlar ile ikinci defa çağırdığımızda orjinal pikseller geri getirilir çünkü iki NOT operasyonu birbirklerini feshederler.

NOT kullanmanın avantajı kodlanmasının kolay olması ve kaplanan alanların bir kopyasının tutulmasına gerek kalmamasıdır. Yanlız bu yaklaşım her yerde kullanılamaz. Mesela, büyütme dikdörtgeni yerine yazı yazmış olsak NOT operasyonu sonrasında yazıyı okumak zor olabilir. Yine NOT operasyonu her zaman net bir tezat oertaya çıkarmaz; örneğin orta koyuluktaki gri renk NOT operasyonundan sonra aşağı yukarı aynı rengini muhafaza eder. Bir diğer meselede NOT operasyonunun Mac OS X desteklenmiş olmamasıdır.

Bir diğer yaklaşımda büyütme dikdörtgenini "hareket eden" noktalı çizgiler ile oluşturmaktır. Bu yaklaşım genellikle resim manipüle etme proğramlarında kullanılır çünkü resimde hani renkler bulunursa bulunsun iyi bir tezad oluşturmak mümkündür. Bunu Qt altında yapmanı yolu QObject::timerEvent() fonksiyonunu yeniden tanımlayıp

DRAFT

```
void Plotter::refreshPixmap()
{
      pixmap.resize(size());
      pixmap.fill(this, 0, 0);
      QPainter painter(&pixmap, this);
      drawGrid(&painter);
      CustomWidgets drawCurves(&painter);
      update();
}
```

refreshPixmap() fonksiyonu grafiği ekranda görünmeyen bir piksel haritasına çizer ve sonra ekranı güncelleştirir.

Biz piksel haritasının boyutlarına aletin boyutları ile aynı olacak şekilde değiştirip onu aletin silme rengine boyuyoruz. Bu renk paletin koyu kısmıdır bunu sebebi ise Plotter sınıfının yapıcısında setBackgroundMode() fonksiyonuna yapılan çaşrıdır.

Sonra piksel haritasını çizmek için bir QPainter nesnesi oluşturup çizim olayını gerçekleştirmek için drawGrid() ve drawCurves() fonksiyonlarını çağırıyoruz. En sonunda tüm aletin yeniden boyanmasına randevü almak için update() fonksiyonunu çağırıyoruz. Piksel haritası alete paintEvent() fonksiyonu içerisinde kopyalanır (p. 121).

```
void Plotter::drawGrid(QPainter *painter)
{
      QRect rect(Margin, Margin, width() -
            2 * Margin, height() - 2 * Margin);
      PlotSettings settings = zoomStack[curZoom];
      QPen quiteDark = colorGroup().dark().light();
      QPen light = colorGroup().light();
      for (int i = 0; i <= settings.numXTicks; ++i)
      {
            int x = rect.left() + (i * (rect.width() - 1) /
                  settings.numXTicks);
            double label = settings.minX + (i * settings.spanX() /
                  settings.numXTicks);
            painter->setPen(quiteDark);
            painter->drawLine(x, rect.top(), x, rect.bottom());
            painter->setPen(light);
            painter->drawLine(x, rect.bottom(), x, rect.bottom() +
                  5);
            painter->drawText(x - 50, rect.bottom() +
                  5, 100, 15, AlignHCenter | AlignTop,
                  QString::number(label));
      }
      for (int j = 0; j <= settings.numYTicks; ++j)
      {
            int y = rect.bottom() - (j * (rect.height() - 1) /
                  settings.numYTicks);
            double label = settings.minY + (j * settings.spanY() /
                  settings.numYTicks);
            painter->setPen(quiteDark);
            painter->drawLine(rect.left(), y, rect.right(), y);
            painter->setPen(light);
            painter->drawLine(rect.left() - 5, y, rect.left(),y);
            painter->drawText(rect.left() - Margin, y - 10,
                  Margin - 5, 20, AlignRight | AlignVCenter,
                  QString::number(label));
      }
      painter->drawRect(rect);
}
```

drawGrid() eğrilerin ve eksenlerin arkasındaki ızgarayı (grid) çizer.

İlk `for` döngüsü ızgaranın düşey çizgileri ile x eksenin küçük çizgilerini çizer. İkinci for döngüsü ise ızgaranın düşey çizgileri ile y ekseninin küçük çizgilerini çizer. drawText() fonksiypnu her iki eksenin küçük çizgilerine tekabül eden rakamların yazılması/çizilmesi amacı ile kullanılır. drawText() fonksiyonunun nehvi şöyledir:

```
painter.drawText(x, y, w, h, alignment, text);
```

burad (`x, y, w, h`) bir dikdörtgeni tanımlar, `alignment` (hiza) ise metnin bu dikdörtgen

içerisindeki yerini belirler, ve `text` ise yazılması.çizilmesi gereken metindir. the text to draw.

```
void Plotter::drawCurves(QPainter *painter)
{
      static const QColor colorForIds[6] =
            { red, green, blue, cyan, magenta, yellow };
      PlotSettings settings = zoomStack[curZoom];
      QRect rect(Margin, Margin, width()
            - 2 * Margin, height() - 2 * Margin);
      painter->setClipRect(rect.x() + 1, rect.y() + 1,
             rect.width() - 2, rect.height() - 2);
      map<int, CurveData>::const_iterator it = curveMap.begin();
      while (it != curveMap.end())
      {
            int id = (*it).first;
            const CurveData &data = (*it).second;
            int numPoints = 0;
            int maxPoints = data.size() / 2;
            QPointArray points(maxPoints);
            for (int i = 0; i < maxPoints; ++i)
            {
                  double dx = data[2 * i] - settings.minX;
                  double dy = data[2 * i + 1] - settings.minY;
                  double x = rect.left() +
                     (dx * (rect.width()-1) / settings.spanX());
                  double y = rect.bottom() -
                     (dy * (rect.height() - 1)/settings.spanY());
                  if (fabs(x) < 32768 && fabs(y) < 32768)
                  {
                     points[numPoints]= Point((int)x,(int)y);
                     ++numPoints;
                  }
            }
            points.truncate(numPoints);
            painter->setPen(colorForIds[(uint)id % 6]);
            painter->drawPolyline(points);
            ++it;
      }
}
```

drawCurves() fonksiyonu eğrileri ızgaranın üzerine çizer. QPainter in haşiyeyi yani marjini içeren kısımlara boyama yapmaması için setClipRect() fonksiyonunu çağırmak suretiyle eğrilerin çizilmesi gereken yeri boyacıya bildirmek suretiyle işe başlıyoruz.

Daha sonra her bir eğri ve eğriyi oluşturan (x,y) kordinat çiftlerinin her biri için iterasyon yapıyoruz. İteratörün `first` isimli değişkeni eğrinin kimliğini (ID) ve `second` isimli değişkeni ise eğrinin datasını verir.

For döngüsünün en derin kısmı kordinat çiftini çizici (plotter) kordinat sisteminden alet kordinat sistemine çevirir ve data makul sınırlar içerisinde yer alıyorsa `points` depişkeninde muhafaza edervariable. Kullanıcı grafiği çok büyütmeye kalkarsa bu durumda 16-bit signed integers değişkeni tarafından muhafaza edilemeyecek rakamlar ortaya çıkar ve

bazı pencere sistemlerinde yanlış çizimlere yol açabilir.

Bir eğrinin bütün kordinatlarını alet kordinat sistemine çevirdikten sonra, o eğrinin çizileceği kalem rengini, Qt de mevcut olan tanımlı renklerden birisini kullanmak suretiyle, belirliyoruz ve sonra drawPolyline() fonksiyonunu çağırarak eğriyi oşuğturan noktalardan geçen çizgiyi çiziyoruz. İşte bu Plotter sınıfının tamaıdır. Geri kalan PlotSettings. sınıfının bir kaç fonksiyonudur.

```
PlotSettings::PlotSettings()
{
      minX = 0.0;
      maxX = 10.0;
      numXTicks = 5;
      minY = 0.0;
      maxY = 10.0;
      numYTicks = 5;
}
```

PlotSettings sınıfının yapıcısı her iki eksenin sınırlarını 0 ile 10 olarak belirleyip küçük çizgi sayısınıda 5 olarak ayarlar.

```
void PlotSettings::scroll(int dx, int dy)
{
      double stepX = spanX() / numXTicks;
      minX += dx * stepX;
      maxX += dx * stepX;
      double stepY = spanY() / numYTicks;
      minY += dy * stepY;
      maxY += dy * stepY;
}
```

scroll() fonksiyonu minX, maxX, minY ve maxY değişkenlerini artırır yada azaltır ki bu miktar o eksenin küçük çizgileri arasındaki mesafenin belirli bir sayı (`stepX` yada `stepY`) ile çarpılması ile elde edilir. Bu fonksiyon Plotter::keyPressEvent() içerisinde kaydırma (scrolling) işlemini gerşekleştirmek için kullanılır.

```
void PlotSettings::adjust()
{
      adjustAxis(minX, maxX, numXTicks);
      adjustAxis(minY, maxY, numYTicks);
}
```

adjust() fonksiyonu mouseReleaseEvent() tarafından minX, maxX, minY ve maxY değerlerini yuvarlayıp her bir eksen için makul olan küçük çizgilerin sayısını belirler. Hususi (private) adjustAxis() fonksiyonu her bir eksen için ayrı ayrı çaprilir.

```
void PlotSettings::adjustAxis(double &min, double &max, int
&numTicks)
{
      const int MinTicks = 4;
      double grossStep = (max - min) / MinTicks;
      double step = pow(10, floor(log10(grossStep)));
      if (5 * step < grossStep)
            step *= 5;
```

```
            else if (2 * step < grossStep)
                  step *= 2;
            numTicks = (int)(ceil(max / step) - floor(min / step));
            min = floor(min / step) * step;
            max = ceil(max / step) * step;
      }
```

adjustAxis() fonksiyonu `min` ve `max` parametrelerini yuvarlar ve numTicks parametresini hesaplamış olduğu ve o anki [`min`, `max`] sınırları için müsait olan küçük çizgi sayısına eşitler. adjustAxis() fonksiyonu değişkenlerin (minX, maxX, numXTicks, vesaire) kopyalarını değil bizzat kendilerini değiştirmesi gerektiğinden parametreler cont türünde olmadıkları gib referans (bir nebze müşir) olmalıdırlar.

adjustAxis() fonksiyonunun büyük bir bölümü küçük çizgiler arasındaki makul mesafeyi (step[32]) bulmaya çalışır. Eksen üzerinde makul rakamların yer alması için step değişkeninin titiz bir şekilde belirlenmesi gerekir. Mesela, 3.8 gibi bir adım (step) eksen üzerinde 3.8 in katlarının görüntülenmesine sebep olurk. Halbuki biz genelde eksenlerde tam yada yuvarlak sayılarla çalışmaya daha alışkın olduğumuzdan böyle bir durum tuhaflık arzeder. Ondalıklı sayılar ihtiva eden eksenler için makul adım 10^n, 2.10^n, 5.10^n ve benzeri sayılardır.

Önce adımın üst sınrını belirleyecek olan takribi bir adım (gross step) değerini hesaplıyoruz,. Sonra bu değere eşit veya ondan küçük olan 10^n türünde bir rakam buluyoruz. Bunu yapmak için takribi adımın (grossStep) 10 tabanına göre logaritmasını alıyoruz ve sonra bu sayıdan küçük ve ona en yakın tam sayıyı bulup 10 değerini bu sayıya yükseltiyoruz. Mesela, takribi adımın 236 olduğunu varsayalım, bu durumda log 236 = 2.37291...; sonra bu sayıyı 2 değerine yuvarlayı 10^2 = 100 değerini elde ederizki buda 10^n türünde adım için bir adaydır.

Bu sayıyı else ettikten sonra onun vasıtası ile diğer iki adayı hesaplarız: 2.10^n ve 5.10^n. Yukarıdaki öisal için dişer iki dayada 200 ve 500 dür. 500 deüeri grossStep değerinden büyük olduğu için onu kullanamayız. Bilakis 200 değeri 236 değerinden küçük olduğu için bu misalde onu kullanıyoruz.

Adım değerini kullanarak `numTicks`, `min` ve `max` değerlerini bulmak gayet kolaydır. Yeni min değeri, orjinal min değerini yuvarlayıp sonra yeni adımın katlarından ona en yakın olanı olarak belirlenir. Aynı şekilde yeni max değeri, orjinal max değerini yuvarlayıp sonra yeni adımın katlarından ona en yakın olanı olarak belirlenir. Yeni numTicks ise yuvarlatılmış mın ve max deşerleri arasındaki aralıkların sayısıdır. Mesela, min 240 ve max 1184 ise fonksiyon içerisinde ynei limitler [200, 1200] ve küçük çizgilerin sayısı 5 olur.

Burada takdim ettiğimiz algoritma tam anlamı ile en elverişli sonucu vermeyebilir. Daha gelişmiş bir algoritma Paul S. Heckbert tarfından “Nice Numbers for Graph Labels” adlı makalesinde verilmiştir ki bu makale “Graphics Gems” (ISBN 0-12-286166-3) kitabında yayınlandı. Yine faydalı olacağını düşündüğümüz Qt nin her üç ayda bir yayınlanan “Qt Quarterly” dergisindeki “Fast and Flicker-Free” makale ki buna http://doc.trolltech.com/qq/qq06-flicker-free.html web adresinde ulaşılabilir.

32 adım

Bu bölüm bizi kitabın il kısmının sonuna getirdi. Bu bölümde Qt aletlerinin nasıl özelleştirilebileceklerine ve QWidget sınıfını temel alarak nasıl yeni aletler oluşturulabilecepini gördük. İkinci bölümde var olan aletlerden yeni aletlerin nasıl yapılabilecepini görmüştük ve altıncı bölümde biraz daha detaylara ineceğiz.

Bu aşamada Qt altında bir GUI proğramının tamamını yazabilmek için gerekli olan bilgiye sahibiz. İkinci kısımda daha derinlere ineceğiz taki Qt den tam anlamı ile faydalanabilelim.

DRAFT DRAFT DRAFT